TABLE DES MATIÈRES

Table des matières

LISTE DES FIGURES

LISTE DES TABLEAUX

CHAPITRE 1

INTRODUCTION

1.1 CONTEXTE DE RECHERCHE

Aujourd'hui, les jeux vidéo sont devenus une forme universelle de divertissement. En effet, d'après la Société Canadienne de Logiciel de Divertissement et la Société de Logiciel de Divertissement des États-Unis de l'Amérique, environ 58 % du américains de nord jouent aux jeux vidéo. De plus, les hommes jouent une moyenne de 18 heures par semaine et les femmes 6,5 heures par semaine [1].

Un jeu vidéo en ligne est un jeu numérique qui nécessite une connexion active à un réseau. Il s'agit non seulement des jeux sur internet, mais également des jeux en ligne via des consoles, téléphones portables ou réseaux P2P.

De nos jours, les jeux vidéo en ligne sont devenus les plus populaires car ils introduisent un style de jeu où des joueurs peuvent rivaliser avec d'autres. Parmi ces jeux, on distingue *world of warcraft* qui a attiré plus de dix million de joueurs en 2008[5]. Ce type de jeux a connu une croissance rapide au cours des dernières années. En effet, les revenus de *Activision Blizzard*, la compagnie qui a crée *world of warcraft*, ont dépassé 912 millions dollars américain en 2013[19].

Aussi, en 2013 le jeu de tir en ligne *CrossFire* était le premier jeu vidéo en ligne à travers le monde avec un chiffre d'affaires de 957 millions de dollars américains [1]. De plus, d'après la Société Canadienne de Logiciel de Divertissement, actuellement il existe plus que 472 studios de développement de jeux vidéo et plus de 20 400 personnes qui ont un emploi direct dans le secteur du jeu vidéo [3].

Malheureusement, une variété d'exploits est apparue dans le monde de jeu virtuel pour le plaisir, pour la victoire, et pour donner quelques joueurs des avantages injustes. En fait, la communauté de jeux a attiré une forme de tricherie connue sous le nom de bots. La raison pour l'utilisation des bots est que certaines parties du jeu sont répétitives et peuvent être ennuyeuses.

Cependant celles-ci sont nécessaires pour améliorer le caractère et progresser dans le jeu. En effet, les tricheurs détruisent l'équilibre du jeu en épuisant rapidement les ressources de jeu. Les joueurs humains honnêtes (qui n'utilisent pas les bots) peuvent donc se sentir démunis, ils perdent l'intérêt, et finalement ils abandonnent le jeu. Cela pose un problème énorme pour les développeurs de jeux, puisque le nombre de joueurs est directement lié au résultat. En effet, entre 2012 et 2015, *World of Warcraft* a perdu presque trois millions d'abonnées [20] ce qui a impacté les revenus de jeu.

Afin de résoudre ce problème, les chercheurs essaient de détecter automatiquement les bots dans les jeux vidéo multi-joueurs en ligne. En effet, ils ont appliqué certaines approches pour la détection automatique des bots. Par exemple : détection de bots en utilisant les preuves humaines [2], ou en analysant la trajectoire [4], ou les actions répétitives [5],...

Ces approches sont toutes efficaces et utiles. Cependant les chercheurs, dans leurs travaux, ont choisi d'approfondir leurs recherches sur certains critères en négligeant d'autres. Par exemple : *Steven Gianvecchio* a choisi de travailler sur les preuves humaines en utilisant un système d'observation des preuves humaines [2]. Par contre, il a négligé les actions répétitives [5].

Comme idée de projet nous avons choisi d'analyser les traces humaines ainsi celles de bots, et de les classifier en se basant sur différentes expériences (plusieurs critères). En utilisant les résultats trouvés, nous allons construire un modèle complet pour un joueur humain et un autre modèle complet pour le bot. Puis, en utilisant une classification supervisée nous allons classifier les agents en joueurs humains et en bots.

1.2 LES BOTS

Un bot (diminutif de robot) est un programme automatisé qui joue le jeu au nom de joueurs humains. Les bots peuvent jouer sans interruption. Ils sont considérés comme la forme d'exploitation la plus commune et la plus difficile à déjouer [2]. Ils ont touché plus les jeux en ligne multi-joueurs tels que *World of Warcraft* [5,2], *Second Life* [6,2] et *Ultima Online* [7,2] et certains jeux en ligne qui ne sont pas multi-joueurs comme *Diablo2* [2]. L'objectif principal des bots de jeu est d'amasser la monnaie du jeu, des articles, et de l'expérience. Ainsi, l'utilisation de bots dans les jeux est un problème grave pour non seulement donner à quelques joueurs des avantages injustes, mais aussi pour la création d'importants déséquilibres dans les économies de jeux. Aussi, la présence de bots peut conduire à des niveaux significatifs de l'inflation dans l'économie du jeu, car ils créent des quantités inhabituelles d'or en tuant des monstres sans fin. Le montant élevé de l'or en circulation augmente les prix des produits, jusqu'à ce que les joueurs honnêtes ne puissent plus les payer. En outre, les bots occupent souvent de bonnes zones agricoles et tuent d'autres joueurs. Cela a un effet indésirable grave

sur l'expérience au jour le jour des jeux pour les joueurs honnêtes et impacte leur motivation au point de les faire cesser de jouer, ce qui a un impact très réel sur les revenus de la société de jeux.

Les premiers bots de jeux ont été développés pour la première génération des jeux en ligne multi-joueurs telle que, *Ultima Online* [2]. Même à cette époque, les opérateurs de bots étaient déjà assez sophistiqués, créant de petites fermes de serveurs pour exécuter leurs bots. Au début, la plupart des programmeurs de bots écrivaient leurs propres clients de jeu. Cependant, en tant que contre-mesure, les sociétés de jeux mettaient fréquemment à jour les jeux, et faisaient une rupture des opérations de ces clients de jeux personnalisés ce qui rend la tâche plus difficile pour les programmeurs de bots qui ont été obligés de mettre à jour leurs script, suivant la dernière version du jeu. Cependant, la complexité des clients de jeu n'a cessé de croître, ce qui rend de plus en plus difficile de développer et de maintenir un client de jeu autonome personnalisé.

Suite à l'augmentation de l'utilisation des bots dans les jeux vidéo en ligne, les concepteurs de jeux vidéo ont perdu des milliers de dollars puisqu'il y a beaucoup de joueurs déçus qui ont cessé de participer aux jeux. Aussi, les concepteurs de jeux ont perdu des revenus dans les ressources utilisées pour prévenir les différentes formes de tricherie, y compris l'utilisation de bots [2].

Il existe de nombreux bots, mais tous peuvent être classés dans l'une des trois grandes catégories:

-Les bots visant à améliorer les capacités intellectuelles d'un utilisateur telles que les programmes de jeu d'échecs, qui peuvent être consultés pour vaincre un adversaire humain. Autrement dit, ils sont en mesure de vaincre le joueur honnête.

-Les bots visant à améliorer les capacités motrices d'un utilisateur telles que la coordination œil-main. Un exemple : un *aimbot* utilisé dans les jeux de *First Person Shooter* pour augmenter les réflexes de l'utilisateur au point de perfection.

-Il existe aussi un grand nombre de bots conçus pour automatiser les tâches répétitives fastidieuses comme la collecte des ressources dans des jeux comme *World of Warcraft*.

Le bot planeur est le bot le plus populaire pour *World of Warcraft* [2], il a été créé par les industries *MDY* et fonctionne simultanément avec le client du jeu, mais nécessite des privilèges d'administrateur système. Ces privilèges permettent au bot planeur de contourner le système anti-bot *Warden*, et ils permettent d'accéder à l'information interne du client de jeu par l'intermédiaire du l'inter-processus-adresse-espace lecture (il peut lire le contenu de l'espace où il est, il prend des décisions et effectue des modifications). Il fonctionne en utilisant un «profil», un ensemble de configurations, y compris plusieurs *waypoints* (coordonnées de la carte dans le monde du jeu) et les options, telles que les niveaux de monstres à combattre. En fonctionnement, le bot contrôle l'avatar

afin de se déplacer entre les *waypoints* donnés, de rechercher et de combattre des monstres qui correspondent aux critères donnés, et de récupérer des objets bonus après avoir remporté les combats.

1.3 LES JEUX VIDÉO MULTI-JOUEURS EN LIGNE

Le programme de collecte de données d'entrée est une version modifiée de l'entrée des enregistrements de l'utilisateur (*Recording User Input*). Il fonctionne concurremment avec le jeu, le vote et l'enregistrement de l'état de l'appareil. Chaque événement d'entrée, tels que le changement de la touche pressée ou la position du curseur, est enregistré avec horodatage (*timestamp*) par rapport à l'heure de début de l'enregistrement.

Dans le monde du jeu, un joueur contrôle un personnage virtuel (avatar). Aussi il peut personnaliser le caractère par l'apprentissage de compétences et l'achat d'articles (tels que des armures, des armes et même les animaux domestiques) avec la monnaie virtuelle. Chaque activité de jeu oblige le joueur à interagir avec le jeu d'une façon différente.

Bien que les humains et les bots interagissent tous deux avec le jeu via la souris et le clavier, ils le font de façon très différente. Ainsi les bots peuvent analyser le graphique du jeu malgré qu'il s'agisse d'un calcul coûteux. Pour minimiser ce coût de calcul, les bots tentent si possible d'obtenir les informations nécessaires, telles que les emplacements de l'avatar, monstres et autres personnages, et les propriétés (santé, niveau, etc.) de l'avatar, par la lecture de la mémoire de programme de jeu. Et de façon générale, les bots contrôlent l'avatar en simulant l'entrée à partir de périphériques via l'API d'appels de l'OS, tels que la mise en état d'appuyer sur la touche ou repositionner le curseur de la souris. Les techniques utilisées par les bots sont souvent brutes, mais dans la plupart des cas, elles sont efficaces.

1.4 Contribution du mémoire

Ce projet fait appel à plusieurs notions et plusieurs domaines : jeu vidéo, sécurité et le forage de données. En effet, nous avons exploité les notions de forage de données afin de proposer une solution pour établir une détection automatique des bots dans les jeux vidéo multi-joueurs en ligne.

Quant à notre contribution, elle consiste à trouver une nouvelle solution pour détecter automatiquement les bots dans les jeux vidéo multi-joueurs en ligne en utilisant les techniques de forage de données. Pour ce faire, une analyse des traces de jeu a été nécessaire. Nous avons fait des expériences pour trouver les différences entre le comportement d'un joueur humain et celui d'un bot. Puis, nous avons construit nos bases de données qui contiennent les résultats de nos expériences.

Ensuite, nous avons classé les traces en se basant à chaque fois sur un seul critère en appliquant une classification supervisée et une classification non supervisée.

Puisque ces résultats sont utiles, pour la détection des bots dans les jeux vidéo en lignes, nous avons créé un modèle de comportement pour les joueurs humains et un modèle de comportement pour les bots. Selon ces modèles nous avons classé les agents qui jouent et nous avons identifié les bots, en détectant ceux qui ont des comportements similaires aux comportements des bots et dissimilaires avec celui des joueurs humains. Ici la détection de bots a été basée sur un modèle complet d'agent (un agent= un bot ou un joueur), c'est-à-dire un ensemble de critères et non pas un seul critère.

1.5 Méthodologie de la recherche

Ce travail de recherche s'est déroulé en plusieurs étapes. La première étape consistait à bien comprendre l'environnement de jeux vidéo multi-joueurs en ligne, le fonctionnement de bots de jeux, ainsi que les méthodes existantes pour détecter les bots dans les jeux vidéo multi-joueurs en ligne. C'est pour cette raison que nous avons commencé notre travail par une étude détaillée sur les bots, le format de traces de jeu, les méthodes existantes pour la détection automatique de bots dans les jeux vidéo en ligne et la prévention des bots.

Puisque nous avions dès le début l'idée de résoudre le problème de détection de bots dans les jeux vidéo multi-joueurs en ligne en utilisant le forage de données, une autre étude est faite sur le processus de l'extraction des données utiles dans une bases de données, le forage de données (il fait partie de processus de l'extraction de données) ainsi que la classification supervisée et la classification non supervisée. Lors de cette deuxième étape, le forage de données et ses opérations sont bien expliqués.

La troisième étape a pour but de trouver une nouvelle approche et l'appliquer. Celle-ci est composée de plusieurs phases. La première phase consistait à analyser les traces qu'on a (durée totale, vitesse, distance, temps de repos) et à comparer entre ces traces. Lors de la deuxième phase, nous avons travaillé sur les traces une par une et sur des intervalles de 200 s et 500 s. Nous avons implémenté une application java que nous avons utilisée pour nos différentes expériences pour détecter les différences entre le comportement d'un joueur humain et un bot. Les résultats des différentes expériences étaient stockés sous des fichiers CSV.

Quant à la troisième phase de la troisième étape, nous avons convertit nos fichiers CSV en fichiers ARFF pour les utiliser sous *weka* [8,9]. *Weka* est un logiciel de forage de données qui implante des algorithmes pour l'analyse de base de données et pour établir une classification

supervisée ou non supervisée. Dans notre projet, les fichiers ARFF sont nos bases de données. Donc nous avons fait une classification supervisée et une classification non supervisée de nos données ce qui nous a donné deux classes : classe bot et classe joueur humain. Nous avons appliqué la classification de données à chaque fin d'une expérience et en se basant sur le critère sur lequel nous avions travaillé. Plusieurs algorithmes de classification supervisée et de classification non supervisée ont été utilisés.

Une étude comparative des résultats obtenus montre bien les différences entre le comportement d'un joueur humain et un bot.

Puis, au cours de la phase 4, nous avons analysé la manière de se déplacer pour les agents.

Comme dernière étape, nous avons réunis les résultats des expériences obtenus et nous avons construit un model complet pour un joueur humain et un autre modèle complet pour les bots ce que nous a permis de classifier les agents non seulement en se basant sur un seul critère mais en se basant sur un modèle complet d'agent.

1.6 Organisation du mémoire

Ce mémoire est organisé en 5 chapitres qui décrivent les étapes décrites dans la méthodologie de recherche.

Le premier chapitre porte sur l'introduction de projet. Il présente les objectifs et les buts de notre projet. Et met en contexte les raisons de cette recherche. Aussi, il définit les bots de jeux vidéo en ligne, leurs avantages injustes et leurs inconvénients sur le monde de jeux et sur son économie. De plus, nous décrivons les caractéristiques de jeux vidéo multi-joueurs en ligne. Aussi, il décrit les objectifs et les étapes de la réalisation de la recherche. Cette étape décrit avec plus de détails les étapes de notre projet.

Le deuxième chapitre portera sur le forage de données (*Data mining*). En effet, nous commencerons par décrire en détails les étapes de processus de l'extraction de connaissances de base de données. Puis nous décrirons en détails le forage de données et les différents domaines où l'utilisation de forage de données a donné des résultats satisfaisants. Ensuite, nous allons présenter les opérations de forage de données: classification supervisée et classification non supervisée.

Le chapitre 3 présentera les différentes méthodes existantes utilisées pour la détection automatique des bots dans les jeux vidéo multi-joueurs en ligne. Au début, nous décrirons le format des traces de jeu [10] et l'utilisation de traces humaines par les bots [2]. Puis, nous présenterons les différentes approches appliquées pour détecter les bots qui consistent à la détection des bots en

utilisant les preuves humaines [2], en utilisant les actions répétitives [5], en se basant sur l'analyse de la trajectoire [4], en utilisant le réseau de neurones bayésien [12], en utilisant l'approche d'analyse de trafic [15] et en utilisant les tests continus intégrés non interactifs [16]. Finalement, nous décrirons la méthode utilisée pour la prévention contre les bots [17].

Le chapitre 4 décrira en détail l'approche que nous proposons pour répondre à la problématique de détection de bots dans les jeux vidéo multi-joueurs en ligne. Nous commencerons par analyser des traces de jeu. Ensuite, nous présenterons nos expériences, les résultats de ces expériences, ainsi que les résultats de classification supervisée ou non supervisée. Enfin, nous décrirons le model de joueur humain et le model de bot qui représentent le résultat final de nos expérimentations. Et nous décrirons les déférences de la façon de se déplacer entre les bots et les joueurs humains.

Enfin, ce travail se termine par une conclusion générale qui fera retour sur tout le projet. En effet, nous décrirons les objectifs réalisés au cours de ce projet. Nous proposerons aussi des améliorations pour ce projet. Nous parlerons des limites pour ce projet. Le mémoire se termine par un bilan personnel de la recherche.

CHAPITRE 2

LE FORAGE DE DONNEES (DATA MINING)

2.1 INTRODUCTION

Au cours des dernières années, la taille des bases de données n'a pas cessé d'augmenter ce qui rend l'extraction de l'information plus difficile. La méthode traditionnelle de l'extraction de connaissances repose sur l'analyse et l'interprétation manuelle. Par exemple, dans l'industrie des soins de santé, il est courant pour les spécialistes d'analyser périodiquement les tendances actuelles et les changements dans les données de soins de santé, sur une base trimestrielle. Les spécialistes fournissent ensuite un rapport détaillant l'analyse de l'organisation des soins de santé. Ce rapport devient la base des futures prises de décision et de la planification de la gestion des soins de santé.

Cependant, cette approche classique d'analyse des données repose fondamentalement sur un ou plusieurs analystes qui doivent être familiarisés avec les données et doivent servir d'interface entre les données, les utilisateurs et les produits. Cette méthode d'extraction de données est lente, coûteuse et très subjective pour plusieurs domaines. En fait, bien que le volume de données augmente de façon très rapide, ce type d'analyse manuelle des données devient totalement impraticable.

C'est pour cela que les chercheurs et les informaticiens ont décidé de trouver une méthode d'extraction de données plus efficace. En effet, durant les années 60, les spécialistes ont découvert une nouvelle approche d'extraction de données, que l'on appelle le forage de données ou le *Data mining.* Cette approche est appelée aussi fouille de données et consiste à l'extraction de connaissances intéressantes à partir d'une base de données.

Au cours de ce chapitre, nous allons commencer par définir le processus de l'extraction de données, le forage de données, ses utilisations dans le monde réel, et finalement les opérations réalisées par le forage de données.

2.2 FORAGE DE DONNES ET EXTRACTION DE DONNEES

2.2.1 PROCESSUS DE L'EXTRACTION DE CONNAISSANCES

Le processus de l'extraction de connaissances dans les bases de données, implique l'utilisation de la base de données ainsi que toute sélection, le prétraitement, le sous-échantillonnage, les transformations, l'application de méthodes de forage de données pour énumérer des classes de celle-ci; et l'évaluation des produits de l'exploitation de données pour identifier le sous-ensemble des motifs énumérés jugés connaissances.

Voilà un schéma détaillant le processus de l'extraction des informations utiles de la base de données :

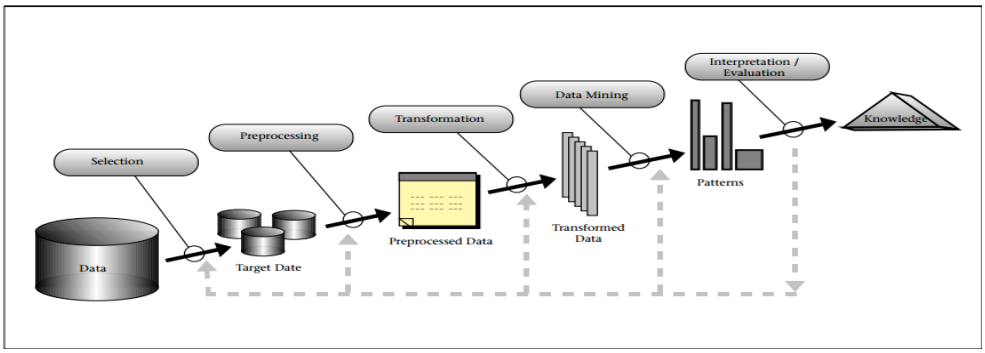

Figure 1: Processus de l'extraction des informations utiles de la base de données [11]

Le processus de l'extraction de connaissances des bases de données est un processus interactif et itératif impliquant de nombreuses étapes avec de nombreuses décisions prises par l'utilisateur. Brachman et Anand (1996) ont donné une vue pratique de ce processus, qui souligne sa nature interactive [11]:

-étape 1 : développement d'une compréhension du domaine d'application et de connaissances préalables pertinentes et identification de l'objectif du processus de l'extraction de données du point de vue du client.

-étape 2 : création d'un ensemble de données de cible à sélectionner un ensemble de données, ou se concentrant sur un sous-ensemble de variables ou des échantillons de données, sur laquelle une découverte va être faite.

-étape 3 : nettoyage des données et de prétraitement. Opérations de base comprennent la suppression du bruit, le cas échéant, la collecte des informations nécessaires pour la modélisation des classes de données.

-étape 4 : réduction des données et de projection: trouver les fonctionnalités utiles pour représenter les données en fonction de l'objectif de la tâche. Avec la réduction de la dimensionnalité

ou des méthodes de transformation, le nombre effectif de variables considérées peut être réduit, ou des représentations invariantes pour les données peuvent être trouvées.

-étape 5 : fixer les objectifs du processus de l'extraction de connaissances pour une méthode d'exploration de données particulière. Par exemple : la synthèse, la classification, la régression, le regroupement, et ainsi de suite…

-étape 6 : choix de l'algorithme d'extraction de données et sélectionner la méthode qui doit être utilisée pour la recherche de modèles de données. En effet, on détermine les paramètres des modèles qui pourraient être appropriés et correspondants à une méthode d'exploration de données particulière avec les critères globaux du processus de l'extraction de connaissances.

-étape 7 : Forage de données (*Data mining*) : rechercher les classes (les modèles) suite à l'application de règles de classification, de *clustering*, de régression,… L'utilisateur peut aider de façon significative la méthode d'exploration de données en effectuant correctement les étapes précédentes.

-étape 8 : interpréter les classes extraites.

-étape 9 : utiliser les connaissances extraites directement, en les intégrant dans un autre système, en les utilisant dans nos prochaines recherches ou tout simplement les documenter ou les utiliser dans nos rapports.

2.2.2 LE FORAGE DE DONNEES (DATA MINING)

Le forage de données est une étape, dans le processus de l'extraction de connaissances des bases de données, qui consiste à appliquer l'analyse de données et la découverte des algorithmes qui produisent une énumération particulière de classes (ou modèles) sur les données. C'est un terme ambigu qui a été utilisé pour désigner le processus de trouver des informations intéressantes dans de grands dépôts de données. Il consiste à trouver les modèles intéressants (les classes) dans les données qui ne font pas explicitement partie des données [11]. Il s'agit de l'application de l'ensemble des méthodes et des algorithmes spécifiques pour l'exploration et l'analyse de (souvent) grandes bases de données; en vue de détecter dans ces données des règles, des associations, des tendances inconnues, des structures restituant l'essentiel de l'information utile… et on parle de connaissances afin de faciliter la prise de décisions.

Par exemple, dans les banques, afin de détecter les fraudes et le vol de cartes de crédit, les analystes appliquent les techniques de forage de données. En effet, selon les données historiques recueillies, ils construisent un modèle de comportement frauduleux. Puis ils élaborent la base de données des comportements, et en appliquant les techniques avancées de forage de données ils détectent les comportements frauduleux similaires.

Le grand l'intérêt actuel pour le forage de données et l'extraction de données est le résultat de succès de nombreuses applications d'extraction de données d'intérêt médiatique. Par exemple, les articles d'intervention pendant les dernières années dans Business *Week, Newsweek, Byte, PC Week,* et d'autres périodiques à large diffusion [11]. Donc le forage de données est utilisé pour des taches d'analyse très complexes, et pour l'extraction des informations très importantes.

Parmi les domaines, où le forage de données et l'extraction de donnée ont été utiles, on peut citer :

Le marketing: en effet, dans le marketing, la principale application est le système de commercialisation de bases de données, qui analysent les bases de données de clients pour identifier les différents groupes de clients et prévoir leur comportement.

L'investissement: De nombreuses entreprises utilisent l'exploration de données (*data mining*) pour l'investissement.

La détection de fraudes : Les systèmes *HNC Falcon* [11] et *Nestor PRISM* [11] sont utilisés pour la surveillance de la fraude de cartes de crédit, surveillant des millions de comptes. Le système *SIAF* [11], est utilisé pour identifier les transactions financières qui pourraient indiquer une activité de blanchiment d'argent.

La fabrication : Le système de dépannage *CASSIOPEE* [11], développé dans le cadre d'un joint-venture entre *General Electric et Snecma* [11], a été appliqué par trois grandes compagnies aériennes européennes pour diagnostiquer et prédire les problèmes pour .0le *Boeing 737*. Pour déterminer les classes de défauts, les méthodes de classification ont été utilisées. *CASSIOPEE* a reçu le premier prix européen pour des applications innovantes.

Les télécommunications: L'analyseur de télécommunications alarme-séquence (*TASA*) [11] a été construit en collaboration avec un fabricant d'équipements de télécommunications et de trois réseaux de téléphonie (*Mannila, Toivonen, et Verkamp 1995*). Le système utilise un cadre novateur pour localiser fréquemment les épisodes d'alarme à partir du flux d'alarme et de les présenter comme des règles. Les grands ensembles de règles découvertes peuvent être explorées avec des outils flexibles de recherche d'informations à l'appui de l'interactivité et l'itération. De cette façon, *TASA* offre l'élagage, le regroupement et les outils de commande pour affiner les résultats de recherche de base d'une force brute pour les règles.

Le nettoyage de données : Le système *Merge-PURGE* [11] a été appliqué à l'identification des demandes d'aide sociale en double (*Hernandez et Stolfo 1995*). Il a été utilisé avec succès sur les données du Département de la protection de l'État de Washington [11].

2.3 LES OPERATIONS DE FORAGE DE DONNEES (DATA MINING) (utilisées dans le projet)

L'objectif de l'application des algorithmes de forage de données est de classifier les données et l'extraction de classes correspondantes aux caractéristiques disponibles. Par exemple : classification des données selon leurs vitesses.

2.3.1 LA CLASSIFICATION SUPERVISEE

La classification supervisée permet de prédire si une donnée (un individu) est membre d'un groupe ou une classe donnée [18]. En effet, lors d'une classification supervisée, les classes sont connues à l'avance (exemple : classe 1 : le bot, classe 2 : le joueur humain), et on dispose des exemples pour chaque classe. En fait, chaque instance est supposée appartenir à une classe prédéfinie.

La classification supervisée est composée de deux étapes. La première étape consiste à construire le modèle à partir de l'ensemble d'apprentissage. L'ensemble d'apprentissage est l'ensemble de couples d'entrées-sorties selon lesquels on va créer notre modèle et on va définir nos classes (groupes).

Par exemple : si vitesse < 10 ➔ classe : joueur humain

Si vitesse > 10 ➔ classe : bot

Puis, la deuxième étape consiste à l'utilisation du modèle qu'on a construit, pour classifier les nouvelles données et pour tester la précision de ce modèle. Pour ce faire, on calcule le taux de classification (taux des instances classées correctement), et le taux d'erreur (= les instances classées incorrectement).

Taux de classification= le nombre des instances classées correctement/ nombre total des instances.

Taux d'erreur = nombre de instances classées incorrectement/ nombre total des instances

Certains algorithmes de classification sont bien connus. Par exemple, la classification bayésienne, les arbres de décision, les réseaux de neurones, les algorithmes génétiques…

Prenons l'exemple des arbres de décisions :

L'arbre de décision est la représentation graphique d'une procédure de classification.

Tableau 1: Exemple d'un ensemble de données d'apprentissage [18]

Ensemble d'apprentissage

Outlook	Temperature	Humidity	Windy	Class
sunny	hot	high	false	N
sunny	hot	high	true	N
overcast	hot	high	false	P
rain	mild	high	false	P
rain	cool	normal	false	P
rain	cool	normal	true	N
overcast	cool	normal	true	P
sunny	mild	high	false	N
sunny	cool	normal	false	P
rain	mild	normal	false	P
sunny	mild	normal	true	P
overcast	mild	high	true	P
overcast	hot	normal	false	P
rain	mild	high	true	N

Jouer au tennis ?

Donc là nous avons deux classes : Classe 1 : N = négatif ; Classe 2 : P= positif

Voilà l'arbre de décision qu'on a obtenu :

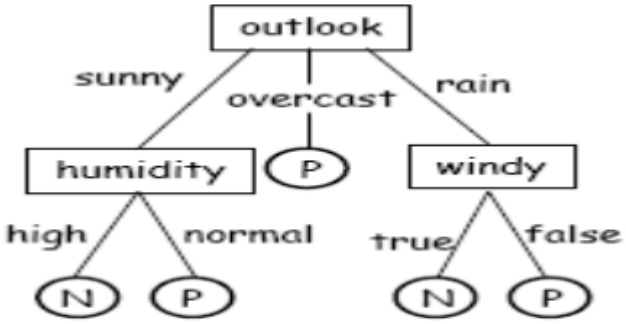

Figure 2 : Exemple d'arbre de décision [18]

Donc d'après l'arbre qu'on a obtenu, on constate que :

*Si le ciel est nuageux (overcast) ➔ classe P ➔ donc on peut jouer au tennis

*Si le ciel est ensoleillé (sunny), on a deux sous classes :

Si l'humidité est élevée (higt) ➔ classe N➔ on ne peut pas jouer

Sinon si l'humidité est normale (normal)➔ classe P ➔ on peut jouer

*Si le ciel pleut (rain), on a deux sous classes :

S'il y'a du vent (true) ➔ classe N ➔ on ne peut pas joueur au tennis

S'il n'y a pas du vent (false)➔ classe P ➔ on peut jouer au tennis

2.3.2 LA CLASSIFICATION NON SUPERVISEE : LE CLUSTERING

Le *clustering* consiste à identifier un ensemble fini de catégories (clusters) pour décrire les données. Chaque membre d'un cluster devrait être très similaire à d'autres membres de son groupe et

dissemblables à d'autres groupes. Les techniques pour la création de grappes comprennent le partitionnement (souvent en utilisant l'algorithme k-means) et les méthodes hiérarchiques (c'est-à-dire chaque sous-ensemble est fractionné en un certain nombre de sous-ensembles), ainsi que le quadrillage [18].

Lors de la classification non supervisée, les classes ne sont pas connues à l'avance. Donc à la fin de la classification non supervisées, les données se classifient dans les classes (les clusters) générées par la classification non supervisée. On obtient : cluster 1, cluster2,…

En appliquant les techniques de classification non supervisée sur l'exemple utilisé dans la classification supervisée, c'est-à-dire sans avoir les noms de clusters, il va donner les même résultats saufs qu'il va appeler les clusters en cluster 1, cluster 2, au lieu de N et P.

2.4 CONCLUSION

Ce chapitre avait pour objectif de définir l'extraction de données, le forage de données ainsi que la classification supervisée et la classification non supervisée. Ce sont les approches que nous avons utilisées dans notre projet afin de détecter les bots dans les jeux vidéo en ligne. En effet, nous avons construit à chaque fois un modèle pour un joueur humain et un modèle pour un bot d'après l'ensemble d'apprentissage qu'on a, puis nous avons classé le reste de donnés selon ces modèles. Ce qui nous a permis de déterminer pour chaque trace s'il s'agit d'une trace humaine ou une trace d'un bot.

Le forage de données a été utilisé depuis plusieurs années dans différents domaines tes que le marketing, la détection de fraudes et la fabrication et voilà nous on l'applique dans un nouveau domaine : la sécurité de jeux vidéo multi-joueurs en ligne, puisque on est convaincu de l'efficacité de cette approche dans plusieurs domaines. Donc, on a été encouragé de l'appliquer pour détecter les fraudes dans les jeux vidéo multi-joueurs en ligne ou ce qu'on appelle les bots.

CHAPITRE 3

LES APPROCHES EXISTANTES POUR LA DETECTION DES BOTS DANS LES JEUX VIDEO MULTI-JOUEURS EN LIGNE

Afin de détecter les bots dans les jeux vidéo multi-joueurs en ligne, les chercheurs ont appliqués plusieurs méthodes. Au cours de ce chapitre, nous commençons par présenter le format de trace de jeu ainsi que la manière d'archivage de traces de jeux. Suite à ça, nous expliquons comment les concepteurs de bots ont utilisé les traces humaines afin d'apparaitre comme des joueurs humains. Finalement, nous présentons les méthodes et les approches utilisées jusqu'à présent, qui ont comme but : la détection des bots dans les jeux vidéo multi-joueurs en ligne.

3.1 ARCHIVE DE TRACE DE JEU

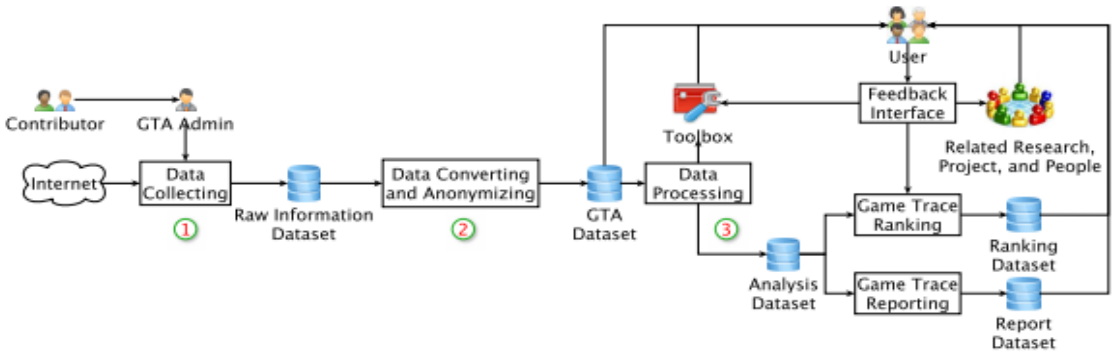

Figure 3 : L'archive de trace de jeu [10]

L'archive de jeu comporte 3 éléments principaux :

Le contributeur : qui est le propriétaire légal de traces de jeu, offre ses traces à l'archive de trace de jeu et permet l'accès du public aux traces.

L'administrateur : aide les contributeurs à ajouter et à convertir des traces de jeu à l'archive de traces de jeu et gère le traitement et le partage de ces traces.

L'utilisateur : accède à des traces de jeu archivés, utilise la boîte à outils de traitement, et obtient les informations de la recherche pertinente à travers l'archive de traces de jeu.

On identifie trois exigences principales pour construire une archive de traces de jeu : La Collecte de traces de jeu, conversion de traces et anonymisation et le traitement de traces.

3.1.1 EXIGENCE1 : COLLECTE DE TRACES

Pour améliorer la lisibilité des traces de jeu et pour faciliter leur échange, un format unifié doit être fourni pour collecter les traces de jeu provenant de diverses sources. Tout d'abord, le format de trace de jeu doit être en mesure d'inclure de nombreux types d'informations sur l'exécution du jeu. Deuxièmement, les formats existent déjà pour des informations spécifiques de jeu. Enfin, en raison de l'évolution rapide de l'industrie du jeu, de nombreux nouveaux jeux peuvent émerger et il serait intéressant des traces avec un formalisme préexistant. Le format doit donc être extensible pour recueillir des traces de futurs jeux, sans affecter l'utilisation de traces qui ont déjà été enregistrées dans l'archive.

Dans la conception de l'archive de trace de jeu, le module *Data collecting* est conçu pour la collecte des traces de jeu à partir de plusieurs sources: les contributeurs (c'est-à-dire les sociétés collaboratrices dans le domaine de jeux vidéo), les données de jeu référentiels partagés publiquement (c'est-à-dire les données publiées par les programmeurs de jeux vidéo), les sites de jeux (ce sont les sites dédiés aux jeux vidéo), etc. Ces traces ont leurs propres formats et certaines d'entre elles peuvent contenir des informations sensibles tels que nombre de relations entre les joueurs, nombre de joueurs, les caractéristiques de jeu...

3.1.2 EXIGENCE 2: CONVERSION DE TRACES ET ANONYMISATION

Le contenu brut de traces de jeu est complexe. Cependant, pendant la procédure de conversion de traces, une petite partie du contenu peut ne pas être liées au jeu, à cause de l'endroit où les traces de jeu sont recueillies. Par exemple, lors de l'exploration de trace des réseaux en ligne Méta-Gaming à partir de leurs sites Web, les informations de publicité dans chaque site peuvent également être recueillies dans la trace. Ainsi, pendant la procédure de conversion des traces de jeu au format unifié de trace de jeu, ce contenu inutile devrait être filtré.

Dans la conception de l'archive de trace de jeu, le module *Data Converting and Anonymizing* est chargé de convertir les traces de jeu de leurs propres formats vers le format unifié de trace de jeu. Ce module peut aussi être chargé de l'anonymisation des informations sensibles.

3.1.3 EXIGENCE 3 : TRAITEMENT DE TRACES

L'archivage doit fournir une boîte à outils pour traiter les traces de jeu converties et générer des rapports couramment utilisés et comprenant les caractéristiques de traces de jeu (par exemple, la taille de trace, le nombre d'éléments d'information). En outre, la boîte à outils pourrait également être utilisée par les utilisateurs de l'archive pour construire des outils d'analyse complets.

Dans le module de traitement des données, une boîte à outils d'analyse des traces complète *Data processing* est fournie. La boîte à outils est composée des informations générales, telles que la taille de la trace, la période, le nombre de relations, et le nombre de joueurs. Aussi, la boite à outils contient les caractéristiques intérieures de jeu, telles que le temps actif de jeu, la durée moyenne d'une session, et le nombre de matchs joués ainsi les métriques de graphes relationnels, tels que le diamètre, la diversité et le coefficient de *clustering*. Ces outils peuvent être utilisés pour construire d'autres outils de traitement et peuvent également être utilisés pour traiter de grands graphiques à grande échelle dans d'autres domaines, tels que les réseaux sociaux et le marketing viral.

3.2 LE FORMAT DE TRACE DE JEU

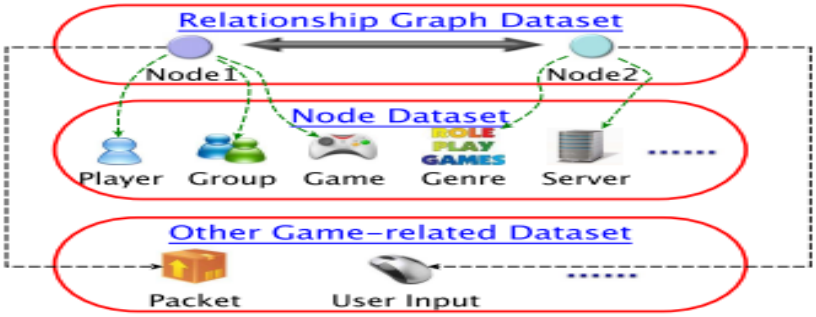

Figure 4 : Format de trace [10]

La trace de jeu consiste en trois ensembles de données :

-l'ensemble de données de relation graphique

-l'ensemble de données de nœud

- l'autre ensemble de données lié au jeu

3.2.1 L'ensemble de données de la relation graphique (Relationship graph dataset)

Cet ensemble contient trois sous-ensembles de données : des informations de base de bord, avec des informations détaillées de bord, et avec des méta-informations.

Les informations de base de bord, comportent l'essentiel des éléments pour les relations (exemple : le bord, type de nœud..).

Les informations détaillées de bord servent pour stocker des autres diverses informations liées au bord. Elles comportent deux parties : la partie fixe et la partie étendue. La partie fixe stocke des propriétés typiques des bords (par exemple l'horodate, durée de vie du bord…). La partie étendue stocke les longs attributs du bord qui ne sont pas communs pour toutes les relations. La conception de la partie prolongée permet de couvrir de nouveaux types d'informations liées au bord.

Les méta-informations comportent la description des ensembles de données de la relation graphique et les définitions de tous les types de bords et de nœuds.

3.2.2 L'ensemble de données de nœud (Node Dataset)

Cet ensemble est destiné pour inclure des informations détaillées de nœud. Pour stocker ces informations dans un format unifié, tout d'abord on les classe. On divise les informations complexes de nœud en des informations statiques (constantes avec le temps, par exemple, la date de naissances) et des informations dynamiques (changeables avec le temps, par exemple, le bord..). Pour chaque nœud, on divise les informations en : informations typiques statiques, informations typiques dynamiques, informations statiques étendues et informations dynamiques étendues. Par conséquence, le nœud peut stocker les informations de toute sorte dans un format unifié.

3.2.3 L'autre ensemble de données liées au jeu (Other Game-Related Dataset)

Cet ensemble sert pour stocker les données de jeux traditionnels, tels que les paquets entre les clients et les serveurs de jeu, lecteur clic flux, etc... On utilise les formats standards de facto et on sauvegarde les données dans l'autre ensemble de données liées au jeu.

Voilà un exemple de traces présentées sous forme le forme de trace de jeu présenté ci-dessous :

Tableau 2: Exemple de traces de jeu [10]

Trace	Période	Size(GB)	# Nœuds (K)	# Liens (M)	Genre de jeu
KGS	2000/02-2009-/03	2	832	27.4	Board
FICS	1997/11-2011/09	62	362	142.6	Board
BBO	2009/11-2009/12	2	206	13.9	Card
XFire	2008/05-2011/12	58	7.734	34.7	OMGN
Dota-League	2006/05-2011/03	23	61	3.7	RTS
DotAlicious	2010/04-2012/02	1	64	0.6	RTS
Doa Garena	2009/09-2010/05	5.2	310	0.1	RTS
WoWAH	2006/01-2009/10	1	91	N/A	MMORPG
RS	2007/08-2008/02	1	0.1	N/A	MMORPG

Board = jeu de table (Exemple : la Monopoli); **Card** = jeu de carte; **OMGN**= réseau de jeux multi-joueurs en lignes; **RTS**= jeux de stratégie en temps réel; **MMORPG**= jeu de rôle multi-joueurs en ligne

Ce tableau résume neuf traces de jeu, formatés dans le format de la trace de jeu. Ces traces ont été recueillies à partir de cinq types de jeux vidéo en ligne multi-joueurs, y compris les jeux de table, les jeux de cartes…

Les traces des jeux *KGS* et *FICS* comprennent un grand nombre de matches en deux jeux de table populaires : *Go* et *Échec*.

Les traces de jeu *BBO* sont recueillies auprès d'un des plus grands sites, où les gens peuvent jouer au *BRIDGE* en ligne gratuitement.

L'ensemble de données de jeu en ligne *XFire* contient les informations de milliers de jeux et des millions de joueurs, ainsi que les relations complexes entre les jeux et les joueurs.

La taille dans ce tableau est la taille des données brutes de chaque trace. Le nœud peut être joueur, avatar, ou serveur de jeu, correspondant à chaque trace. Dans *XFire*, on utilise le lien de l'amitié. Pour les autres traces, on définit le lien que de jouer dans un même match.

Finalement, les traces de jeu *WoWah* et *RS* ont été recueillies auprès de sites de jeux vidéo multi-joueurs en ligne les plus populaires.

3.3 UTILISATION DE TRACES HUMAINES PAR LES BOTS

3.3.1 NAVIGATION DE BOTS VIA LA LECTURE DES TRACES HUMAINES

L'imitation de traces humaines a été utilisée pour synthétiser et détecter les primitives de mouvement dans les jeux. Et ici l'objectif des bots est d'apparaître comme des humains, ainsi idéalement leur comportement doit être impossible à distinguer de ce qui est attendu de celui d'un joueur humain dans une situation donnée.

Dans ce contexte, on souligne *Unreal Tournament 2004.* Ce tournoi comprend un système de script intégré qui utilise un script irréel. Le script irréel est un langage de programmation qui fournit une API pour le moteur *UnrealEngine* et est utilisé par des packages de haut niveau. Ce script est utilisé aussi par *GameBots2004* pour permettre des programmes externes de contrôler les bots joueurs et de recevoir des informations sur leur état. Il permet également l'enregistrement des traces de jeu et des jeux joués par les humains et les bots. Ainsi, l'interface de *GameBots2004* permet à un programme de contrôler un bot à l'intérieur du l'*Ureal Tournament* en utilisant une connexion de socket réseau. Les messages sont échangés synchrones et asynchrones entre l'agent et le bot de connexion de *GameBots2004* fonctionnant sur le serveur de *Ureal Tournament,* ce qui permet à l'agent de recevoir des mises à jour sur le jeu et l'état de bot et d'envoyer des commandes de contrôle de son mouvement et ses actions [14].

Le bot *UT2* utilise le Contrôleur de Trace Humaines (HTC) afin de détecter et de récupérer les problèmes de navigation rapidement, tels que ne pas apparaitre humain lors du suivi intégré dans le graphique de navigation. Dans la compétition *BotPrize 2010,* le bot utilise le HTC afin d'améliorer son comportement de navigation. En effet, lorsque le bot se coince tout en se déplaçant le long d'un chemin ou pendant un combat, il exécute les actions sélectionnées par le contrôleur. Le HTC utilise une base de données de jeux humains déjà enregistrés pour exécuter un comportement de navigation similaire à celui des joueurs humains. En effet, le *mod BotPrize* a été décompilé dans son script original en utilisant les outils standards qui viennent avec *Unreal Tournament 2004*.

Deux types de points de données sont enregistrés dans la base de données pour chaque joueur humain: les données de pose et les données d'événement. La pose du joueur comprend sa position courante, son orientation, sa vitesse et son accélération. Ces données sont enregistrées à chaque cycle logique, soit environ dix fois par seconde. Les événements du joueur sont enregistrés avec leur horodatage et incluent les mesures prises par les joueurs ou différents types d'interactions avec d'autres joueurs ou l'environnement.

Trois types de jeux sont enregistrés:

Les jeux standards : capturent le comportement des joueurs humains en jouant premier joueur tireur de la variante norme de la mort de match *Unreal Tournament 2004*.

Les jeux de jugement : sont enregistrés en vue de surmonter les différences de comportement potentielles qui se manifestent lorsque les joueurs humains jugent également.

Les jeux de couverture synthétiques : en effet, des données synthétiques ont été recueillies en instruisant des joueurs humains à volontairement passer du temps à naviguer dans les zones à faible couverture de données.

Afin d'être en mesure de récupérer rapidement les traces pertinentes de joueur humain, un schéma d'indexation efficace des données est nécessaire. Les deux régimes les plus efficaces d'indexation sont : Indexation basée sur *Octree* et Indexation basée sur la navigation graphique.

a) INDEXATION BASEE SUR OCTREE

Un *octree* est une structure de données couramment utilisée en infographie et vision à l'index des informations spatiales. Il est construit en trouvant le milieu d'un ensemble de points géométrique, et la subdivision des points en huit sous-ensembles définis par les trois axes alignés hyperplan passant par le point. Le processus se poursuit de manière récursive jusqu'à ce qu'une condition de cessation (telle que la profondeur, la plus petite dimension de la feuille…) soit satisfaite. Par exemple, afin d'indexer les données de la pose dans la base de données de traces de jeu de joueur humain, un *octree* est construit sur l'ensemble de tous les points avec une condition de terminaison (exemple : atteindre le plus petit rayon, atteindre le nombre minimum des points…). Puis chaque point de la base est marqué par une feuille d'*octree*. Compte tenu de l'emplacement de bot, il est possible de faire une indexation basée sur *octree* et de trouver le plus petit nœud *octree* [14].

b) INDEXATION BASEE SUR NAVIGATION GRAPHIQUE

Cette méthode d'indexation consiste à relier un certain nombre de sommets nommés répartis dans le niveau avec des bords d'accessibilité. En fait, les nœuds du graphe de navigation peuvent être utilisés comme étiquettes pour les points de la base de données de trace, ce qui permet au contrôleur de récupérer rapidement les points qui sont les plus proches d'un nœud particulier. Une fois les points les plus proches à l'emplacement de bot sont trouvés, la base de données donne l'ensemble des points dans la base de données de pose qui sont le plus proche du point [14].

Prenons par exemple le cas de Colisée :

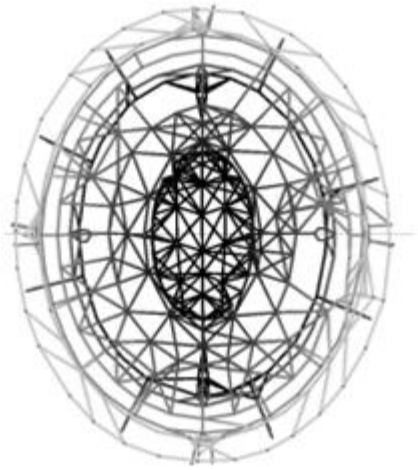

Figure 5 : Graphe de navigation pour le colisée [14]

Dans ce cas, les niveaux de l'*Unreal Tournament* et de nouveaux autres jeux similaires viennent des graphes de navigation qui relient un certain nombre de sommets nommés répartis dans le niveau avec des bords d'accessibilité.

Un problème fondamental auquel sont confrontés tous les concepteurs du comportement humain (concepteurs de bots) est le problème de correspondance, c'est à dire la différence entre les actions et les observations disponibles pour les humains et ceux offerts aux agents artificiels. Ce problème peut comprendre des différences dans la fréquence de décision, la nature et la quantité des informations exprimées dans les capteurs, les différences de morphologie du corps ou de la capacité et ainsi de suite. Ce problème conduit parfois à l'incapacité du bot à reproduire une réaction humaine, soit parce que ses observations sont insuffisantes ou parce que ses actions sont limitatives.

Les caractéristiques environnementales constituent une autre classe de problèmes à faire avec des caractéristiques particulières de l'environnement. En effet, puisque les dangers de l'eau sont particulièrement difficiles pour le contrôleur de trace humaine, au cours de la compétition, le bot *UT2* a utilisé un contrôleur d'eau spécialement conçu quand il était coincé dans les zones d'eau. Bien que cette solution partielle ait amélioré la performance du bot sur les niveaux avec des obstacles d'eau, il ne résulte pas dans un comportement particulièrement similaire à celui d'un humain.

Afin d'évaluer la contribution de Contrôleur de Trace Humaine (HTC), des comparaisons ont été faites entre les trois versions du bot, où la seule différence était le contrôleur utilisé.

Les trois contrôleurs suivants ont été utilisés dans la comparaison:

• Le contrôleur *Null* ignore tout simplement la condition coincé et continue à tout ce que l'action mettrait fin.

• Le contrôleur *Unstuck script* est un contrôleur script conçu pour obtenir le bot *Unstuck* durant l'évaluation.

• Le contrôleur de trace humaine tel qu'il a utilisé pendant la compétition ***BotPrize2010***

Suite à cette comparaison, on obtient :

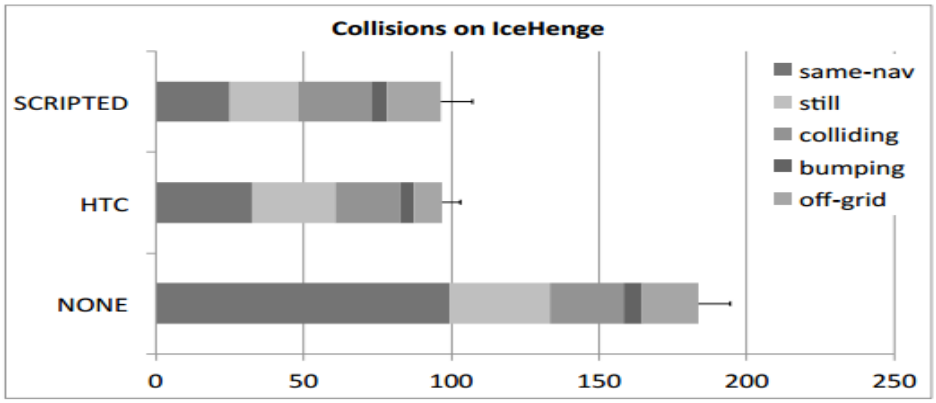

Figure 6: Résultat de comparaison les trois contrôleurs (NULL, UnStack, BotPrise) [14]

Le travail démontre la faisabilité de l'utilisation de grandes bases de données de comportement humain pour soutenir la prise de décision en ligne. L'approche peut évoluer et améliorer l'expérience acquise naturellement avec des experts du domaine; elle est applicable à l'objectif explicite de la construction de comportement de type humain; et il soutient l'imitation, une primitive qui est omniprésente dans les exemples de comportement intelligent dans la nature.

3.3.2 PROGRAMMATION DES AGENTS JOUEURS DE FOOT EN MODELESANT LE COMPORTEMENT HUMAIN

L'apprentissage automatique (*Machine Learning)*, un des champs de l'intelligence artificielle, qui permet de créer de la connaissance de manière automatique à partir des données brutes. L'apprentissage automatique permet de développer des systèmes de reconnaissance vocale, de détection de fraudes… Il peut servir aussi à la conception des bots. En effet, les joueurs humains peuvent apprendre à jouer de nombreux jeux, vite et bien. Donc, il est logique de tenter de les imiter et de transférer leur expérience aux agents informatiques.

Dans le cas du simulateur de Robot de football [13], des programmeurs ont utilisé la programmation génétique pour évoluer une équipe complète d'agents, tandis que d'autres proposent une architecture d'agents où l'apprentissage peut être utilisé à de nombreux états de jeu (réactif ou tactique ou stratégique), et acquérir les compétences nécessaires pour l'adaptation à l'adversaire.

Le joueur humain interagit avec une interface ***Gui soccerclient*** qui permet de jouer le Robot Football comme un jeu vidéo. Cette interface graphique envoie les commandes humaines à l'interface

Soccerserver et affiche l'état du champ à l'humain. Puis, le formateur ***Traner*** est utilisé pour définir les règles du jeu dans un état particulier, parce que de nombreux états différents sont nécessaires pour apprendre des modèles généraux**.**

De l'interface graphique, une trace est obtenue en observant le jeu humain. Les enregistrements sont obtenus pour chaque cycle de serveur. Cette trace est faite de plusieurs couples (S, A), où **S** est l'observation faite par les capteurs de l'agent et **A** est l'action menée par le joueur humain dans cette situation.

Puis, des techniques d'apprentissage machine sont utilisées pour obtenir un classificateur qui détermine quelle action doit être prise dans une situation particulière. Ensuite, le classificateur sera traduit en code C, qui sera utilisé pour contrôler un agent de football. Si le processus de modélisation est correct, l'agent de football jouera Robot football d'une manière similaire à un joueur humain.

L'interface robot Football affiche une vue en temps réel 2D complète du champ, tout comme le moniteur de football. Des positions absolues sont calculées au moyen de calculs trigonométriques, permettent d'obtenir la position absolue des objets à partir de la position connue des bannières répartis le long de la bordure du champ.

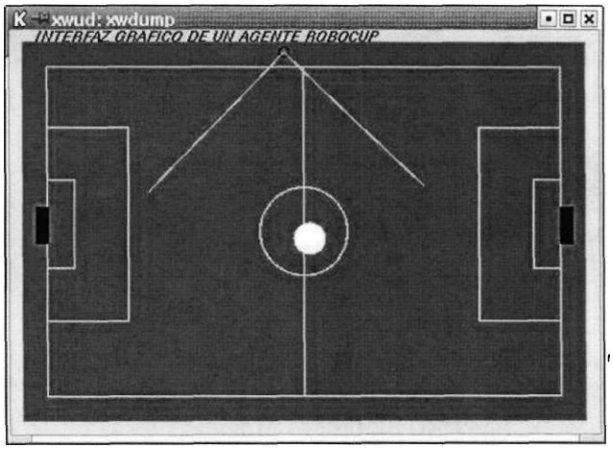

Figure 7: L'interface graphique de Robosoccer [13]

Bien que tout le champ soit visible, seuls les objets à l'intérieur du cône de vision de l'agent sont affichés. En outre, dans le robot de football, la perception des objets éloignés est bruyante. Ainsi, les objets sont affichés sous forme de cercles de probabilité: le rayon du cercle dépend du rayon de l'objet et de distance de l'objet.

Les commandes autorisées par l'interface sont:

Tournez à gauche: le joueur peut tourner le corps de l'agent à 10 ° vers la gauche.

Exécuter lent / rapide: seulement ces deux types de bord sont autorisés.

Coup de pied balle: envoie le ballon en direction de vue la ligne de l'agent avec une probabilité de succès de 60%.

Coup de pied à l'objectif: envoie le ballon en vers le but, avec une probabilité de succès de 99%.

Afin d'avoir un ensemble diversifié de cas pour des comportements d'apprentissage, un ensemble diversifié de situations doit être présenté au joueur humain. L'agent formateur a été utilisé à cette fin. Il permet de positionner l'agent, la balle, et même l'équipe opposée dans des positions arbitraires dans le domaine. De cette façon, un positionnement initialement défensif de l'équipe adverse est atteint.

Dans les situations les plus complexes, le joueur humain jouera au sein d'une équipe contre une équipe complète de face. Dans le cas de robot football les chercheurs ont utilisé une situation simple où un seul agent humain joue contre une équipe défensive. Cette équipe est basée sur des zones, où chacun des membres de l'équipe se trouve localisé. Parmi les agents, le joueur le plus proche de la balle prend le rôle de chef de la file. Le reste des agents maintiennent une distance avec le chef de file de manière à maintenir une formation. Les agents déterminent indépendamment qui est le leader et transmettent cette information aux autres agents. Lorsqu'un agent se déplace loin de sa zone, il tente à passer la balle à un autre agent, si possible. Sinon, il continue vers le but afin de marquer. Le gardien de but suit un comportement similaire: il reste dans sa zone et cherche la balle.

Ensuite, le comportement humain peut être modélisé à bien des égards. Certains d'entre eux ont été traditionnellement utilisés en IA (Intelligence Artificielle) : règles, des arbres, des modèles de régression, des programmes logiques, etc.

Dribble des adversaires statiques implique un attaquant qui avance avec le ballon et marque, après avoir les agents adversaires statiques situés près du but. Ces opposants ne peuvent frapper la balle quand elle est très proche pour eux.

Suite à l'expérience effectuée sur le robot de Football, on constate que dans certains cas, le joueur affiche les comportements erronés suivants:

-L'agent tente de botter le ballon quand il est trop loin, ou lorsque l'angle n'est pas approprié.

-Lorsque l'agent est très proche de la balle, il tourne à gauche **continuellement**.

-L'agent entre en collision avec un adversaire et arrête là.

Une fois que l'agent effectue ces comportements défectueux, il ne sort jamais de ces états. Ces comportements se produisent dans une situation où le seul agent qui peut initier des actions (et changer le monde) est l'attaquant. Quand il y a des opposants plus actifs dans le domaine, la situation va changer indépendamment de l'agent, et l'agent va sortir de ces états statiques plus facilement.

Le math avec les adversaires est le comportement le plus complexe appris: un attaquant doit se rendre à la balle et marquer contre trois défenseurs et un gardien de but. Pour cette tâche, il semble qu'il serait souhaitable de choisir des adversaires très *diíBcults* (c'est-à-dire puissants et confiants), comme *CMUnited* [13] qui est constitué d'une équipe de simulateurs qui utilisent les nouvelles techniques multi-agents pour assurer la coordination d'adaptation, ou le simulateur *FC-Portugal* qui a été développé à l'université de Porto en 2000 [13]. Ces deux simulateurs étaient les précédents champions *Robocup*. Cependant, le joueur humain été incapable de les battre. Cela était dû au fait que ces équipes qui jouent très bien, mais aussi au fait que l'interface n'est pas assez sensible. Ce dernier problème ne pouvait être résolu, car le serveur *Robosoccer* n'a pas été conçu avec un jeu interactif à l'esprit.

En résumé, l'interface de robot de football permet à l'utilisateur d'envoyer des commandes de bas niveau (tableau de bord, tour, et coup de pied) à *Soccerserver*. Les cas d'entrée/sortie générés par le joueur humain ont été utilisés par un algorithme d'apprentissage automatique pour apprendre un modèle. Ce modèle a ensuite été introduit dans un agent de l'ordinateur.

3.4 LES TECHNIQUES UTILISEES POUR LA DETECTION DE BOTS

3.4.1 DETECTION DE BOTS EN UTILISANT LES PREUVES HUMAINES

Afin de déterminer les différences de comportement entre les bots et les joueurs humains, les chercheurs ont réalisé une expérience basée sur les preuves interactives humaines, en particulier, les *CAPTCHAs*[2,16]. En effet, Cette méthode permet de détecter les bots en suivant passivement les actions d'entrée qui sont difficiles à effectuer pour les bots (saisir un *CAPTCHA*). Pour ce fait, les chercheurs ont développés un système d'observation de preuves humaines qui analyse les actions saisies par les utilisateurs avec un réseau de neurones (*Neural Network*) en cascade de corrélation pour distinguer les bots des humains.

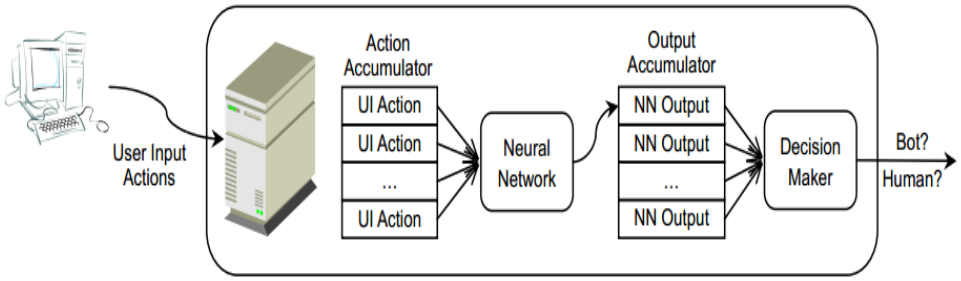

Figure 8: Système d'observation de preuves humaines [2]

Le système d'observation de preuves humaines se compose d'exportateurs côté client et d'un analyseur côté serveur.

Comme chaque client du jeu reçoit déjà des événements saisis par l'utilisateur, l'exportateur côté client utilise simplement les informations disponibles pour dériver des actions d'entrée : frapper, pointer, cliquer, glisser, déposer…, et les envoie vers le serveur avec des données régulières liées au jeu. Idéalement, l'exportateur côté client doit être mis en œuvre en tant que partie intégrante de l'exécutable du jeu ou des systèmes anti-triche existants.

L'analyseur côté serveur est composé de deux éléments principaux: le classificateur de l'action entrée par utilisateur et le décideur.

Pour la classification d'action d'entrée, le classificateur utilise l'algorithme des réseaux de neurones artificiels (Il s'agit d'un algorithme de *data mining*). Le décideur se réfère à l'utilisation de la sortie accumulée à partir du réseau de neurones pour déterminer si les données d'entrée utilisateur correspondant sont susceptibles d'être un bot ou un joueur humain.

Différents algorithmes peuvent être appliqués pour consolider les classifications accumulées. On utilise le régime "droit de vote" : si la majorité de la sortie du réseau de neurones classe les actions saisies par l'utilisateur comme un bot, la décision sera que le jeu est exploité par un bot, et vice versa.

Le processus de décision est un résumé des classifications des actions saisies par l'utilisateur sur une période de temps. Bien que la classification individuelle puisse ne pas être correcte à 100%, plus la sortie est accumulée, plus on a la confiance à la décision prise. D'autre part, la sortie plus accumulée, plus les actions de saisie de l'utilisateur sont nécessaires, ce qui correspond à plus de stockage de données et plus de temps pour la prise de décision. Et puisque les actions entrées par l'utilisateur sont des messages courts, la consommation de bande passante supplémentaire induite par l'exportateur côté client est négligeable. La présence de l'exportateur est donc imperceptible pour les

utilisateurs finaux. Sur le côté serveur, l'évolutivité est essentielle à la réussite du système d'observation par preuves humaines. L'analyseur de côté serveur est très efficace en termes d'utilisation de la mémoire et d'utilisation du processeur. La taille de la mémoire supplémentaire consommée par un joueur est comparable à la taille du nom de l'avatar du joueur. Le noyau du processeur unique est capable de traiter des dizaines de milliers d'utilisateurs simultanément en temps réel.

Afin de déterminer les différences de comportement entre les bots et les joueurs humains, les chercheurs ont réalisé une expérience en se basant sur le jeu *world of Warcraft*. C'est un jeu vidéo en ligne multi-joueurs développé par la société *Blizzard* [5,2]. L'expérience consistait à inviter 30 joueurs humains différents pour jouer *World of Warcraft* et de recueillir 55 heures de leurs traces utilisateur-entrée [2]. De manière correspondante, ils ont fait fonctionner des bots avec 10 profils différents dans 7 endroits dans le monde du jeu pour 40 heures et ils ont recueilli leurs traces d'entrée.

Suite à cette expérience, on constate qu'il y a deux différences majeures entre les bots et les joueurs humains. Tout d'abord, les frappes des bots sont plus rapides que celles des joueurs humains. Ceci est causé par le fait que les joueurs humains doivent entreprendre une action de frappe par le mouvement physique de doigts, et donc, appuyer sur les touches avec une fréquence élevée serait très fatigant. Deuxièmement, les frappes de bot présentent des motifs périodiques évidents.

En effet, les bots présentent deux caractéristiques uniques. Premièrement, contrairement aux joueurs humains, qui déplacent la souris avec une vitesse très variable indépendamment de la longueur de déplacement, les bots ont tendance à déplacer la souris à plusieurs vitesses fixes pour chaque déplacement et cette vitesse augmente linéairement avec la longueur du déplacement. Deuxièmement, les bots font une quantité importante de mouvements à grande vitesse avec le déplacement de zéro, c'est-à-dire qu'après une série de mouvements rapides, le curseur revient exactement à son point d'origine. Un tel comportement est absent dans les données humaines, parce qu'il est physiquement difficile et inutile.

Le système d'observation des preuves humaines a quatre paramètres configurables: le nombre d'actions par bloc, le nombre de nœuds, le seuil, et le nombre de sorties par bloc de sortie.

Avec le réseau de neurones configuré, le seuil et le nombre de sorties par bloc déterminent la performance globale du système. Le seuil peut être augmenté ou diminué de la valeur par défaut de 0,5 pour solliciter le réseau de neurones vers des robots ou des humains, l'amélioration de véritable taux positif ou le vrai taux négatif, respectivement. Le nombre de sorties par bloc affecte à la fois la précision de détection et la vitesse de détection du système.

Puis les chercheurs ont effectué une deuxième expérience de plus petite envergure sur un jeu différent : *Diablo2*, c'est un jeu vidéo d'action et de rôle de type *hack and slash* développé par *Blizzard*. Malgré qu'il ne s'agit pas d'un jeu multi-joueurs, il dispose d'une économie comme les jeux en lignes multi-joueurs et est également tourmenté par des bots de jeu.

Le *MMBot* est un bot libre et populaire pour *Diablo2* qui est construit en utilisant le langage de script *AutoIt*. C'est un script d'automatisation de Windows qui permet d'accéder au contenu de jeu. Similaire à *Glider*(le bot planeur de *World of Warcraft*), *MMBot* automatise diverses tâches dans le jeu pour accumuler des trésors ou d'expérience. Cependant, contrairement à *Glider*, *MMBot* ne lit pas l'espace mémoire du jeu, mais est plutôt entièrement basé sur le clavier/automatisation de la souris, et le balayage par pixel [2].

Suite à cette expérience, on constate que le système d'observation des preuves humaines est capable de capturer certains invariants dans le comportement des bots à travers différents jeux et différentes implémentations de bot, indiquant le potentiel possible des jeux basés sur le système d'observation des preuves humaines.

Le système d'observation de preuves humaines sur le côté du serveur est nécessaire pour traiter des milliers d'utilisateurs simultanément en temps réel. Il est donc efficace en termes de mémoire et de calcul. Ainsi, le temps de calcul est également très faible. En effet, l'analyseur est capable de traiter environ 296 heures de traces par seconde en utilisant un seul processeur. Aussi, un serveur avec 5000 utilisateurs générerait environ 1,38 heure de traces par seconde.

Le système d'observation de preuves humaines appuie sur les différences de comportements de jeu tels que les actions de clavier et de souris entre joueurs humains et les bots de jeu pour identifier les programmes de bot.

Les résultats montrent que le système peut détecter plus de 99% des bots de jeu actuelles sans faux positifs dans une minute et les frais généraux de la détection est négligeable ou mineur en termes de trafic induite du réseau, CPU, et le coût de la mémoire. Donc, on peut conclure que ce système est efficace pour capturer les bots, ce qui soulève la barre contre les exploits de jeu.

3.4.2 DETECTION DE BOTS EN UTILISANT LES ACTIONS REPETITIVES

L'impact des bots sur un jeu peut être substantiel. En particulier, la présence de bots peut conduire à des niveaux significatifs de l'inflation dans l'économie du jeu, car ils créent des quantités inhabituelles d'or par tuer des monstres sans fin.

Warden est une application de monitoring (surveillance) qui protège *World of Warcraft*. Le *Warden* est exécuté sur l'ordinateur du joueur, lorsqu'il est en train de jouer *world of Warcraft* et fait des contrôles sur les programmes suspects tels que débogueurs ou des bots.

Les bots sont conçus pour effectuer des portions simples et répétitives du jeu, soulageant le joueur de ces tâches répétitives. Habituellement, les bots se déplacent dans le monde virtuel, tuant les entités mobiles non-joueur. Pour ce fait, le joueur doit spécifier un certain script qui détermine quelles actions sont choisies par le personnage, en fonction de la situation de jeu. Cela comprend la séquence de sorts (c'est-à-dire les compétences, exemple : frapper, bloquer…) qui sont exprimés dans le combat, les circonstances dans lesquelles le caractère se guérit lui-même, et s'il ramasse le butin de cadavres ennemis. Le joueur met en outre un chemin que le bot doit suivre, et les ennemis doivent l'attaquer. Les bots suivent habituellement plus ou moins exactement les mêmes chemins, mais les laissent parfois pour attaquer un monstre ou recueillir le butin d'un ennemi tué.

En regardant un graphique de mouvement, il est souvent simple pour un humain de reconnaître les schémas de mouvement répété. Toutefois, cette question doit être répondue automatiquement à l'échelle de grandes populations de joueurs.

Ainsi, quand un joueur déplace son personnage dans le monde virtuel, le client du jeu envoie régulièrement des paquets avec les nouvelles coordonnées sur le serveur. Par conséquent, les coordonnées de mouvement sont facilement disponibles sur le serveur.

a) CONSTRUCTION DE LA ROUTE

Pour construire la route, tout d'abord, on reconstruit la route que le personnage avait prise dans le monde du jeu en connectant simplement les points qui arrivent sur le serveur de coordonnées. Ensuite, on traite les points en utilisant l'algorithme *Douglas-Peucker line simplification*, qui sert à simplifier une poly ligne par la suppression de nœud. Selon cet algorithme la poly ligne est simplifiable et remplacée par une ligne simple (deux nœuds) si la distance de son nœud le plus éloigné de la droite formée par les extrémités de la poly ligne est inférieure à un seuil.

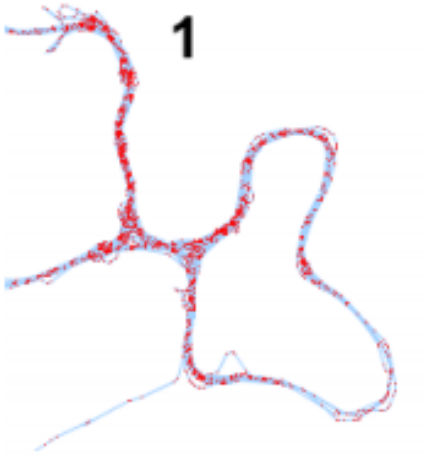

Figure 9: Construction de la route [5]

b) EXTRACTION DE WAYPOINTS

Lorsque le même chemin est pris à plusieurs reprises, des points de différentes pistes sont accumulés à des groupes de points de haute densité. Le *waypoint* est une zone d'un diamètre fixe centrée sur la base d'un cluster. La zone du *waypoint* est considérée comme passée dans chaque course. On utilise l'algorithme de classification non-supervisée *k-means* où la taille de cluster est limitée à la taille du *waypoint*. En effet, on assure qu'un *waypoint* contient toujours tous les points de cluster et que le personnage passe à chaque fois à travers l'un des points de sa région. Enfin, on élimine le chevauchement de *waypoints* en gardant uniquement celui avec le plus grand nombre de points.

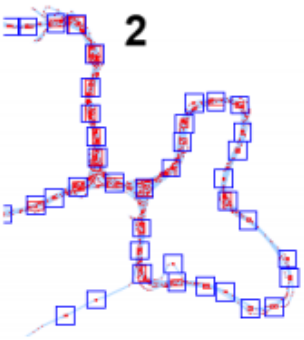

Figure 10 : Extraction de waypoints [5]

Puis, on se retrouve avec un ensemble de *waypoints* répartis sur toute la longueur de la voie que le personnage a pris, un peu clairsemée dans les sections de la voie qui ont été voyagé moins souvent, et plus dense le long d'une "sentiers battus", comme les points de chaque terme s'y accumule. S'il n'y a pas de *waypoint* pour un point, on le passe simplement. Comme chaque *waypoints* est identifié de façon unique, une route peut être décrite par la séquence de points de cheminement.

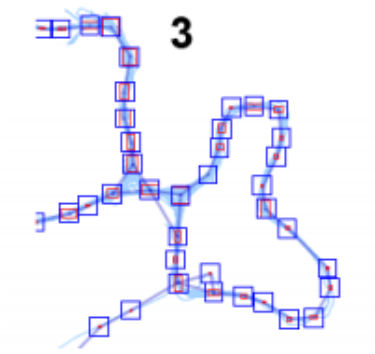

Figure 11: Simplification de chemin [5]

c) TROUVER LES REPITITIONS DANS LA ROUTE

On utilise un tableau des suffixes pour trouver les répétitions dans la séquence de mouvement (*waypoint*). On étend le tableau des suffixes en calculant aussi le tableau du plus long préfixe commun (LCP).

Le tableau de LCP contient le préfixe commun le plus long que les actions d'un suffixe avec la précédente dans le tableau. Dans le cas de jeu *world of warcraft*, la table de LCP permet de repérer rapidement les séquences répétées. En particulier, une longue LCP signifie qu'une longue partie de la route a été répétée. Dans le cas parfait, un bot va toujours passer par les mêmes points de cheminement dans le même ordre, ce qui rend facile de trouver le chemin complet de bot de ces répétitions. Cependant, un bot peut choisir de ne pas courir le chemin complet en une seule fois, ou il peut ne pas suivre le chemin assez exactement à toujours passer tous les *waypoints* sur son chemin.

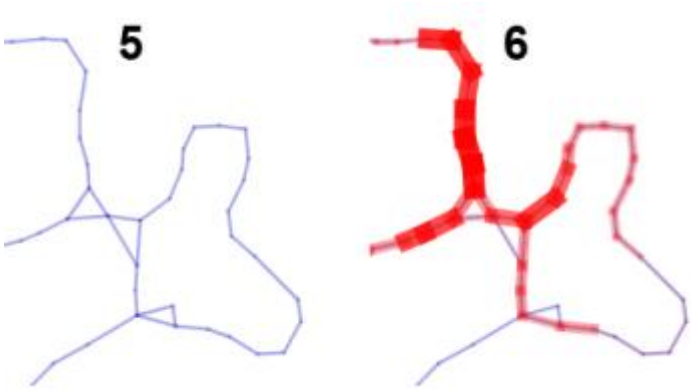

Figure 12 : Chemin plus simplifié : suppression de waypoints [5]

d) METRIQUES DE DETECTION DE BOTS

Décision basée sur la moyenne de nombre de passes par segments de chemin :

Un segment de chemin est une paire de *waypoints* adjacents dans la séquence de mouvement ainsi qu'une ligne droite d'un point de passage jusqu'à ce que le caractère atteigne le suivant. Le

nombre de fois où un segment de trajet est passé indique la fréquence à laquelle un caractère est déplacé sur un segment de voie, indépendamment de la direction. On calcule la moyenne de nombre de passes par le segment de chemin en divisant le nombre total de segment de chemin passe par le nombre de segments de chemin distinctes.

Comme le déplacement sur le même chemin signifie également qu'aucun nouveau *waypoints* n'est créé, le nombre de segments de chemin distincts demeure constant, tandis que le nombre de segments total et le nombre de passes par le segment de chemin augmente régulièrement. Ceci est le cas typique pour les bots.

Dans le cas des joueurs humains, d'autre part, de nouveaux segments de chemin sont créés en permanence, parce que le mouvement aléatoire crée de nouveaux segments entre les anciens *waypoints*, ou encore touche à des zones inexplorées. Ainsi, le nombre de segments de chemin distinct et le nombre total de segments ne cesse d'augmenter. Cela conduit à une faible moyenne de nombre de passes par le segment de chemin, un nombre qui reste constant sur la course du jeu, généralement bien inférieurs de 2 passes par segment de chemin.

En exploitant cette différence, on peut facilement faire la distinction entre les bots et les joueurs humains. En effet, les traces de jeux de bot montrent de façon constante l'augmentation du nombre depuis que le commencement de jeu, alors que le nombre pour les traces de jeu humaines reste à un faible niveau constant.

Décision basée sur des répétitions de la séquence de waypoints :

Les bots prennent toujours les mêmes chemins, et lorsqu'ils ne peuvent pas se déplacer à travers les mêmes coordonnées exactes, leur route contient quand même de nombreuses et longues séquences répétitives par rapport aux routes humaines. Les valeurs de tableau des suffixes LCP de la séquence de *waypoint*, capture exactement cette propriété. On calcule la valeur moyenne de LCP pour assurer que la route doit contenir non seulement longues mais aussi de nombreuses répétitions pour afficher les nombres clairement suspectes.

Les joueurs humains ne sont jamais généralement proches des valeurs de LCP que les bots atteignent, car il est très difficile de garder constamment passer les mêmes points de cheminement dans la même séquence. Le LCP moyen pour le joueur humain se pose sur un niveau faible, tandis que le LCP de bot augmente comme ils visitent le même chemin, encore et encore. Cela fournit un bon discriminateur entre les joueurs humains et les bots.

Une expérience est faite en se basant sur le jeu *world of Warcraft*, on a mis en place le serveur de *world of Warcraft* et les traces de jeu collectées de deux bots différents et dix joueurs humains différents. Pour le serveur, on a modifié l'open source de *ArcEmu* (c'est l'émulateur de serveur) et on a connecté tout le trafic entrant et sortant et des informations supplémentaires sur l'état du jeu en format lisible par le joueur humain dans un fichier texte.

Pour créer des traces de bots, on a exécuté à la fois le bot planeur *Glider* ainsi que le bot libre *ZoloFighter*, en utilisant les mêmes caractères de modèle que les joueurs humains. On a créé deux traces de jeu par bot, en utilisant des chemins différents à chaque fois. Les traces de jeu, obtenues ont été ensuite introduites dans le *WOWalyzer*(C'est un programme Java, qu'on a développé pour afficher et analyser des traces de jeu).

Le mouvement de joueur humain semblait aléatoire, avec très peu de *waypoints* ou des segments de tracé qui ont été passés plus d'une fois. D'autre part, les bots ont rapidement montré des motifs de répétition du mouvement.

Suite à l'application de la méthode de détection de bots basée sur la moyenne de nombre de passes par segments de chemin on obtient :

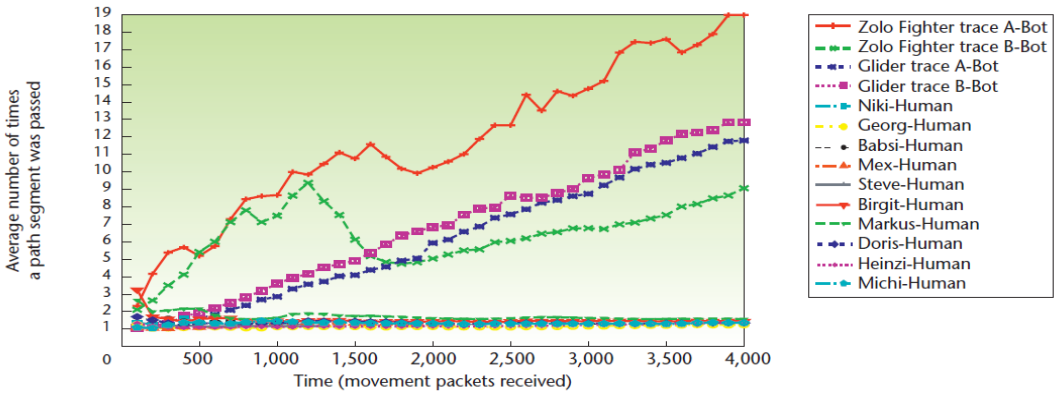

Figure 13: Moyenne de nombre de passes par segments de chemin [5]

Suite à l'application de la méthode de détection de bots basée sur des répétitions de la séquence de waypoints, on obtient :

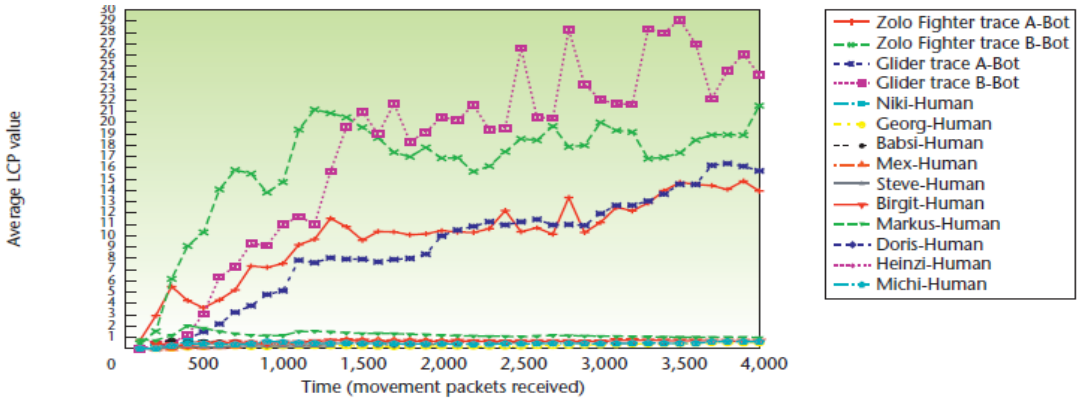

Figure 14: Moyenne de nombre de répétions de la séquence de waypoints [5]

Pour mesurer les séquences répétées dans les séquences de *waypoints*, on a calculé la moyenne de la LCP (tableau des suffixes créé à partir de la séquence de *waypoint)*. Comme la montre la figure ci-dessus, les échantillons de bots montrent des valeurs nettement plus élevées que les échantillons humains, car ils contiennent beaucoup plus et aussi plus de séquences répétées. Selon le graphique, les joueurs humains ne se déplacent dans des motifs répétés, qui a gardé le LCP moyenne sur un faible niveau stable (en dessous de 2, parfois même en dessous de 1).

Suite à cette expérience, on peut distinguer de façon fiable entre les bots et les humains. En général, les chiffres pour les bots et les humains commencent à diverger, car les bots commencent à répéter leur modèle de mouvement sur leur deuxième manche d'un chemin. Une alerte de bot est déclenchée chaque fois qu'un certain seuil est atteint.

Le système actuel met en valeur le comportement idéal pour des mécanismes de détection de bot côté serveur, car il fonctionne de manière totalement transparente pour le joueur. Cela rend difficile pour le joueur de découvrir ce qui est exactement déclenché une interdiction de compte. Toutefois, la découverte du système pourrait certainement conduire à des programmeurs de bots qui tentent de mettre en œuvre des contre-mesures. Alors que les tests préliminaires ont montré que le système mis en place est capable de distinguer de manière fiable entre les bots et les joueurs humains.

3.4.3 DETECTION DE BOTS EN SE BASANT SUR L'ANALYSE DE TRAJECTOIRE

Récemment, des logiciels anti-triche (Exemple : *PunkBuster*, *GameGuard*…) ont été largement déployés dans les jeux en ligne pour empêcher la tricherie. Ces logiciels sont livrés aux clients de jeu, de sorte qu'ils ne peuvent pas être désinstallés, même si le client les a supprimés. Ils fonctionnent en se cachant dans le processus de client de jeu. Ils font le suivi de l'ensemble de l'espace de mémoire virtuelle (pour empêcher la modification des images exécutables du jeu) et bloquent les programmes suspects qui pourraient être des outils de piratage, et bloquants certains appels d'API. Ce type de logiciels peut détecter presque tous les plug-ins des outils qui se fixent à un programme de

client du jeu pour inspecter ou modifier les états de jeu lorsque le jeu est en cours d'exécution. Malheureusement, il ne peut pas arrêter l'utilisation généralisée des bots, y compris, les bots *CRBot, ICE, Eraser…*

C'est pour cela les chercheurs ont proposé une nouvelle méthode de détection des bots dans les jeux vidéo basée sur l'analyse de trajectoire. C'est une méthode qui peut être appliquée sur n'importe quel jeu dans lequel le mouvement de l'avatar est contrôlé par les joueurs directement. En effet, en analysant les comportements cachés dans les trajectoires, on vise à détecter si une entrée invisible est un bot ou un joueur humain.

Dans ce cadre, les chercheurs ont appliqué cette méthode sur le jeu *Quake2*. Il s'agit d'un célèbre jeu de tir à la première personne (*First Person shooter*). Le joueur adopte le rôle d'un caractère particulier et tire ses ennemis via l'interface utilisateur. Ce jeu prend en charge une fonction d'enregistrement de jouabilité qui garde la trace de chaque action et de mouvement, ainsi que le statut de chaque personnage et un objet, tout au long du jeu. Avec une trace enregistrée, on peut reconstruire un jeu et l'examiner de toute position et l'angle désiré avec les opérations de type magnétoscope. Les joueurs utilisent souvent cette fonction pour évaluer leurs stratégies de performance et de combat. Aussi pour cette expérience, on a utilisé trois bots populaires pour le jeu *Quake2 : CRBot, ICE, Eraser.*

Puis on a comparé les trajectoires des avatars de joueurs humains et des bots. Ensuite, on a analysé leurs habitudes de navigation agrégée et leurs trajectoires individuelles.

a) STRUCTURE DE NAVIGATION AGREGEE

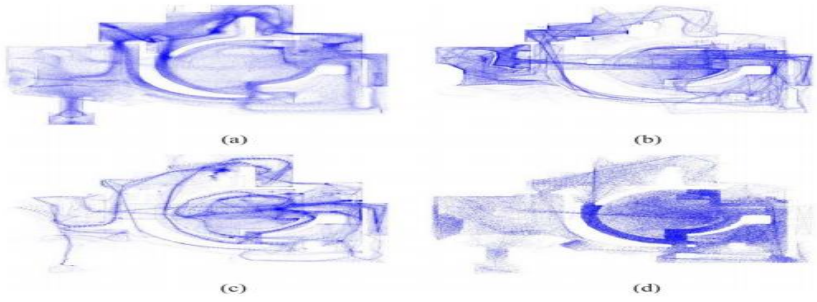

Figure 15 : Navigation agrégée [4]

Dans cette figure, les zones de haute densité sont les endroits que les joueurs visitent plus fréquemment, tandis que les zones clairsemées représentent les bâtiments, d'autres types d'obstacles que les joueurs ne peuvent pas passer, ou seulement des zones que les joueurs ne sont pas intéressés à

visiter. Les figures montrent que le niveau de jeu est formé par des carrés, des places et des ruelles étroites.

Les joueurs humains ont tendance à explorer toutes les zones sur la carte. Ainsi, la figure 15(a) montre le terrain le plus complet du niveau. En revanche, les algorithmes de routage utilisés par les bots peuvent avoir des difficultés à naviguer certains endroits, et évitent donc certaines parties de la carte. Par ailleurs, pour réduire la probabilité d'être attaqués, les joueurs humains évitent normalement les espaces ouverts.

Par conséquent, comme représenté sur la figure 15(a), les joueurs humains évitent la place dans le milieu de la carte, et sont resté dans les ruelles environnantes. Ceci est indiqué par la forte densité de parcelles dans les ruelles. En revanche, les bots restaient souvent sur cette place, sans doute parce qu'il y'a un grand espace et il est facile d'obtenir partout de ce domaine basé sur un algorithme de routage simple. Et même si les joueurs humains passaient la plupart de leur temps dans les zones étroites et les espaces confinés, il y avait de grandes variations dans leurs trajectoires.

Il y a deux raisons pour ce phénomène : tout d'abord, les principales routes sont assez larges, afin que les joueurs se déplacent de façon irrégulière dans l'espace plutôt que de rester au milieu d'un itinéraire. Cela peut être dû aux préférences des joueurs. Ensuite, comme les combats peuvent se produire à tout moment, n'importe où, les joueurs humains se déplacent souvent stratégiquement. En revanche, les bots adoptent des schémas très différents de mouvement sur les routes. Ceci suggère que ces bots ont tendance à suivre les habitudes de déplacement même s'ils se déplacent à travers la même allée que les joueurs humains.

b) LES TRAJECTOIRES INDIVIDUELLES

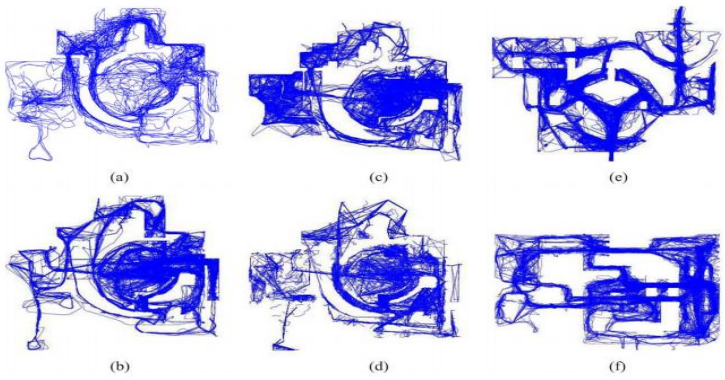

Figure 16 : Trajectoire individuelles [4]

Les trajectoires des joueurs humains contiennent beaucoup plus d'irrégularité et de tours que celles de bots. Il ya deux explications possibles à cela: tout d'abord, les mouvements irréguliers réduisent les chances d'être attaqué par l'arrière. Ensuite, la prise de décision humaine peut être irrégulière et donc ne peut pas être consistante dans le temps. Un joueur humain peut changer sa pensée à tout moment et ajuster le pas et la direction du caractère, et fonder la décision sur des facteurs imprévisibles. En revanche, les trajectoires des bots sont principalement caractérisées par des voies droites et longues.

Les différences entre les modèles de mouvement des joueurs humains et les bots fournissent le cadre conceptuel de l'analyse du comportement et le schéma de détection de trajectoires de bots. Même si les développeurs de bots peuvent contrer l'algorithme de détection par la formation des bots pour imiter le comportement humain, il reste toujours des comportements humains qui sont difficiles à imiter.

L'algorithme de détection de bots comprend trois parties: la mesure de la dissemblance, la représentation de trajectoire, et la détection de bot via la classification.

Compte tenu de la représentation de la trajectoire, on peut utiliser une méthode de classification ou de groupement pour la détection de bot. Pour ce fait on utilise l'algorithme de *data mining*, SVM (*Support Vector Machine*)[4], qui tente de résoudre un problème de minimisation sans contrainte, et l'algorithme KNN (*K nearest neighbour*)[4] pour la plupart des évaluations. Les résultats expérimentaux démontrent que, en termes de performance, KNN est très efficace en termes de complexité de temps par rapport à SVM (ou SSVM). Deuxièmement, l'approche avec *Isomap* (Réduction non linéaire de dimensionnalité) [4] fonctionne mieux qu'une approche sans *Isomap*. Troisièmement, l'approche qui tient compte des informations temporelles, à l'entrée à la longueur de code dérivé par le modèle de chaîne de Markov, surpasse toutes les autres méthodes.

Les méthodes KNN et SVM combinés avec *Isomap* surpassent ceux sans elle, même si le bruit est ajouté aux trajectoires de bots (une stratégie de camouflage commune) pour contrer l'algorithme de détection.

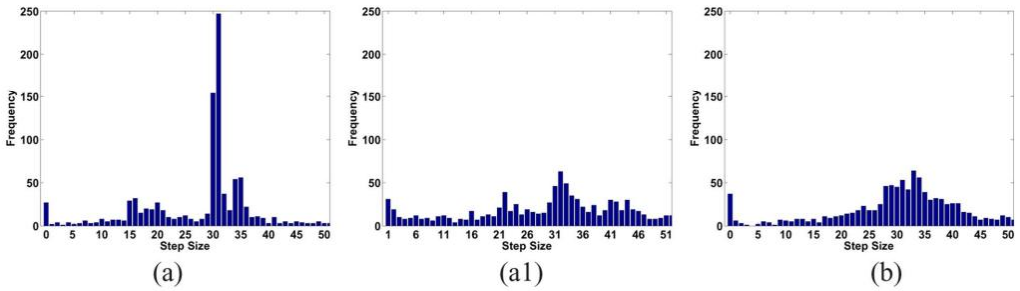

Figure 17 : Isomap [4]

Donc, on peut conclure que cette méthode est efficace pour montrer la différence entre un joueur humain et un bot

3.4.4 DETECTION DE BOTS EN UTILISANT LE RESEAU DE NEURONES BAYESIEN

Bien que de nombreux jeux de *First-Person Shooter* soient désormais protégés par des logiciels anti-triche, la tricherie ne peut être totalement évitée. Outre l'utilisation de la solution logicielle, les sociétés de jeux en ligne ont également besoin d'engager suffisamment de personnes pour surveiller le jeu en permanence afin de découvrir les tricheurs potentiels. Par exemple, certains serveurs de jeu en ligne ont des administrateurs et s'ils découvrent certains joueurs suspects, ou bien s'ils reçoivent un nombre suffisant de plaintes d'autres joueurs contre un joueur, l'administrateur a le droit d'expulser ce joueur de participer au jeu en ligne.

Bien que les tricheurs peuvent être mis en œuvre dans de nombreuses façons différentes et peuvent fonctionner différemment pour atteindre le même but, ces formes de triche produisent essentiellement des habitudes de jeu similaires.

Pour ces raisons, les chercheurs ont décidé de trouver une solution efficace et évolutive pour détecter automatiquement si un joueur est un tricheur potentiel ou non. En effet, on a appliqué l'algorithme de réseau de neurones bayésien dynamique sur un jeu *First-Person Shooter* multi-joueurs en ligne. Il s'agit d'un algorithme de *data mining* qui fait la classification de données en utilisant la technique de l'apprentissage non supervisé.

Voilà le réseau de neurones bayésien utilisé pour détecter les bots :

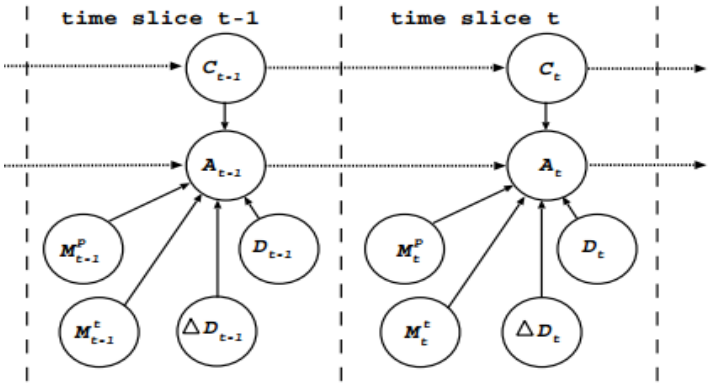

Figure 18: Réseau de neurones utilisé pour détecter les bots [12]

Dans un jeu *First-Person Shooter*, plusieurs informations d'état de joueur influencent certainement sur la précision de son but. Par exemple, plus le joueur est proche de la cible, plus il a la chance de l'atteindre. En outre, il est plus facile de viser une cible statique qu'une cible en mouvement à grande vitesse. De même, un joueur doit avoir une précision de visé quand ce joueur est immobile, comme comparer à un joueur qui est visé tout en se déplaçant dans le même temps. Il est certain que l'utilisation d'un bot de but affectera la précision de but d'un joueur. Par conséquence, la distribution de probabilité de la précision de visée d'un joueur est dépendante selon plusieurs facteurs qui sont indépendants d'autres facteurs :

*si le joueur est un tricheur ou non (noté dans le réseau de neurones par C)

*si le joueur se déplace ou non (noté dans le réseau de neurones par Mp)

*si la cible visée par le joueur se déplace ou non (noté dans le réseau de neurones par Mt)

*si le joueur est en train de changer la direction viséve (noté dans le réseau de neurones par ΔD)

*la distance entre le joueur et la cible visée (notée dans le réseau de neurones par D).

On modélise le processus de la cible visée en tant que processus de *Markov* de premier ordre [12] (le processus de *Markov* est une chaine de variables aléatoires). Une fois qu'un joueur a un but précis, de petits réglages sont nécessaires pour conserver la précision. Cela signifie que la distribution de probabilité de la précision d'un joueur sur une certaine tranche de temps t dépend de la précision du joueur sur la tranche de temps précédente t - 1.

a) EFFICACITE DE RESEAU DE NEURONES BAYESIEN POUR DETECTER LA TRICHERIE

Pour démontrer l'efficacité de la méthode proposée, les chercheurs ont mis en place trois bots de but (un bot de but est celui qui cherche une cible). Lorsque le bot de but est activé, il trouvera la cible la plus proche et la vise avec précision. Il continue à viser cette cible même s'il existe une cible

plus proche, jusqu'à ce que la distance entre le joueur et l'objectif actuel soit supérieure à un certain seuil.

Les chercheurs ont effectué dix sessions distinctes de jeu et ensuite ils ont organisé les données dans trois ensembles de données. Ces ensembles de données sont les suivants:

L'ensemble de données A : formé de trois joueurs honnêtes et trois tricheurs utilisant le bot basique de but. Cet ensemble est utilisé pour la formation du réseau bayésien dynamique.

Ensemble de données B : formé de trois joueurs honnêtes et trois tricheurs utilisant le bot basique de but.

Ensemble de données C : formé de trois joueurs honnêtes, trois tricheurs utilisant le bot de but, de commutation automatique (allumer et éteindre le bot de but en alternance), et trois tricheurs utilisant le bot de but le plus avancé.

Tout d'abord, les chercheurs ont commencé par étudier la possibilité de la détection des tricheurs sans produire aucun faux positif pour les joueurs honnêtes. Pour ce fait, ils ont utilisé l'ensemble de données A en tant que ensemble de formation. Selon la classification fournie par l'ensemble de données A, on déduit pour l'ensemble B la classe de chaque joueur c'est-à-dire on déduit s'il s'agit d'un joueur humain ou d'un bot.

Voilà les résultats de cette expérience :

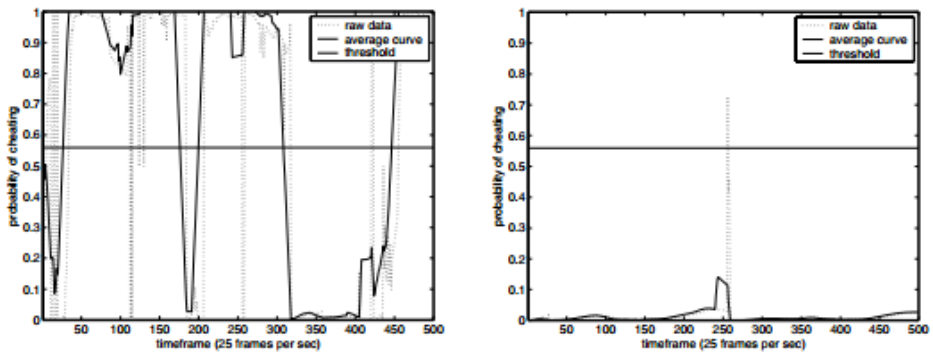

Figure 19: Comparaison de possibilité de tricherie entre un bot de but et un joueur honnête [12]

Suite à cette expérience, on remarque que dans la plupart du temps, la probabilité de tricherie pour un joueur honnête maintient bien en dessous du seuil. Alors que, la probabilité de tricherie pour un tricheur peut varier au-dessus du seuil assez fréquemment, ce qui indique que cette expérience est très efficace pour détecter l'utilisation de bot de but.

Il existe certaines périodes où un tricheur a une faible probabilité de tricherie, cela se produit probablement lorsque le tricheur n'a pas de cible visible à viser.

b) ADAPTABILITE DU COMMUTATION AUTOMATIQUE ET MANQUE DE CIBLE

Les chercheurs ont réalisé une deuxième expérience afin d'étudier la possibilité de détecter les tricheurs qui utilisent l'un des deux bots de buts les plus avancés, ce qui signifie qu'ils se chargent suite à une commutation automatique ou en manquant intentionnellement d'une cible. Pour ce fait on utilise l'ensemble de données A en tant que données de formation et ensuite on déduit les classes pour les données établies en C.

Pour la même raison qu'il n'y a pas de cible visible, il existe des périodes de temps que la probabilité est bien en dessous du seuil. La probabilité ne chute pas même si le bot de but est éteint, cela parce que ce dernier aide le joueur à viser la cible. La probabilité diminue lorsque le bot rate sa cible, cependant, elle augmente à nouveau lorsqu'il vise à sa cible dans des délais plus tard. Ceci suggère que cette méthodologie peut effectivement détecter l'utilisation de bot de but même lorsqu'il a la fonctionnalité avancée de commutation ou des manques intensionnels.

Voilà les résultats de cette expérience :

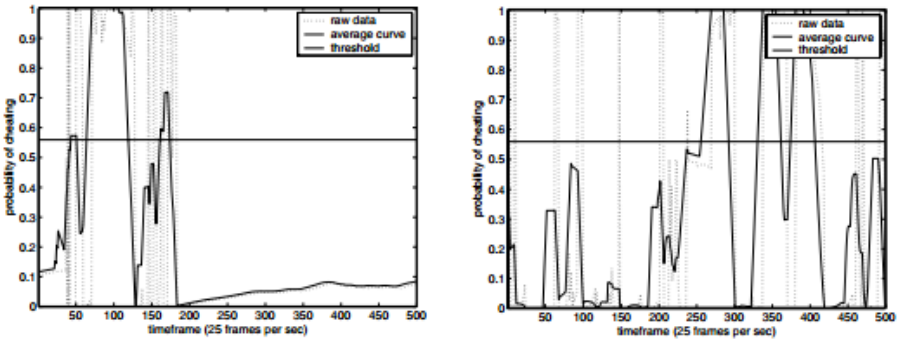

Figure 20: Comparaison de l'adaptabilité à Auto-commutation et aux manques intensionnels entre un bot et un joueur honnête [12]

Enfin, on utilise la notion de validation croisée pour valider les résultats précédents par former et inférer avec différentes combinaisons d'ensembles de données. On forme d'abord avec l'ensemble de données B et on déduit les classes des données définies en C, puis on forme avec l'ensemble de données B et on déduit les classes pour les données de l'ensemble A. on utilise le même seuil pour tous les cas de test pour déterminer si le joueur est un tricheur ou non. Même lorsqu'on utilise des différents ensembles de données pour la formation, la méthodologie est toujours efficace pour déterminer si un joueur utilise le bot de but ou non. La probabilité inférée d'un tricheur fluctue au-dessus du seuil alors que la probabilité inférée d'un joueur honnête est en dessous du seuil.

Les résultats expérimentaux montrent que le réseau bayésien dynamique est une solution efficace et évolutive dans la détection du bot tricheur visant un jeu multi-joueurs en ligne.

3.4.5 DETECTION DE BOTS EN UTILISANT L'APPROCHE D'ANALYSE DE TRAFIC

Dans les jeux en ligne multi-joueurs, déterminer si un caractère est contrôlé par un bot ou non est difficile, car un personnage de bot contrôlé obéit complètement aux règles de jeu. Pour l'identification de bots, une nouvelle méthode consiste à lancer manuellement un dialogue avec un personnage suspect, puisqu'un bot ne peut pas parler comme un humain.

Ragnarok Online, l'un des jeux de rôle en ligne multi-joueurs, les plus populaires au monde. Il a été créé par la Société Gravité de la Corée du Sud en 2002 [15]. *Ragnarok Online* possède des aspects de base plus ou moins standard, pour la plupart des jeux de rôle en lignes multi-joueurs, par exemple, les caractères de formation, l'obtention d'un meilleur équipement…

Figure 21: Imprime écran de Ragnarok Online [15]

Après avoir joué plusieurs parties de *Ragnarok Online*, les personnages deviennent progressivement plus fortes et mieux équipées en gagnant de points d'expérience et en accumulant le butin de combat. Cependant, le combat répété peut devenir un peu ennuyeux. C'est pour cela, certains joueurs cherchent à mettre en place des bots qui peuvent automatiquement et à plusieurs reprises effectuer des tâches assignées sans intervention humaine.

Il existe plus de douzaine de bots pour *Rangharok Online*. *Kore* est le bot le plus connu pour ce jeu en ligne. C'est un programme écrit en Perl et C, indépendant de la plateforme, et open-source. Aussi, un autre bot très populaire pour ce jeu est le *DreamRO*. C'est un leader des serveurs de *Ragnarok Online* qui répond aux besoins de la recherche sociale et de sensations fortes de joueurs.

Kore et *DreamRO* sont deux bots autonomes. Ils peuvent communiquer directement avec les serveurs de jeu sans l'intervention des joueurs humains de *Ragnarok Online*. Leurs actions sont à base de script, et presque toutes les actions disponibles dans le client du jeu sont couvertes. Quand un programme de bot est en marche, il rend compte en permanence les dernières informations et l'état actuel du jeu, exemple : l'emplacement d'un personnage, la dernière action réalisée…

Analyse de trafic

La distinction entre les bots et les joueurs humains apparait au moment de la réalisation de commandes clients. En effet, pour les joueurs humains, des commandes de clients, par exemple, aborder un autre personnage, attaquer un monstre à proximité.., sont déclenchées par des touches de clavier ou clics de souris. Alors que, pour les bots, le déclenchement des commandes du client est décidé par le moteur de décision dans le programme de bot. Ainsi, la conception du moment d'émission de la commande suivante est importante car elle conduit à des contradictions importantes dans les modèles de trafic entre les différentes séries de bot, et entre les bots et les joueurs humains.

Suite à une analyse de la date de libération des commandes du client contre les bots on constate que la libération de commandes, à la fois pour Kore et DreamRO, dépend de l'expiration de temporisateur, et l'arrivage de paquets de données de serveur. L'utilisation de minuteries périodiques est intuitive et raisonnable, étant donné que de nombreuses actions dans un jeu sont de nature itérative.

L'analyse de trafics comprend quatre étapes : Tout d'abord, en se basant sur l'intuition, on inspecte la régularité intégrée dans le trafic. Deuxièmement, on examine la date d'émission des commandes client par rapport à l'heure d'arrivée de la plus récente du paquet de données du serveur. Puis, on assiste à la rafale de trafic des traces de paquets. Enfin, on identifie des schémas particuliers de comportement humain causés par la sensibilité aux conditions du réseau, qui, bien sûr, ne sont pas possédées par les bots.

a) REGULATITE DE TRAFIC CLIENT

Suite à l'expérience réalisée, on remarque que, les traces de joueurs A1et D1 montrent l'aléatoire dans le temps des paquets inter-arrivés. Alors que, les traces de bots *Kore3* et *DreamR0* suggèrent l'existence d'un mécanisme de temporisateur de déclenchement et l'absence du hasard qui caractérise les actions humaines. Donc, on peut déduire que la distribution du temps des paquets inter-arrivés, est plus régulière pour le trafic de bot que le trafic de joueurs.

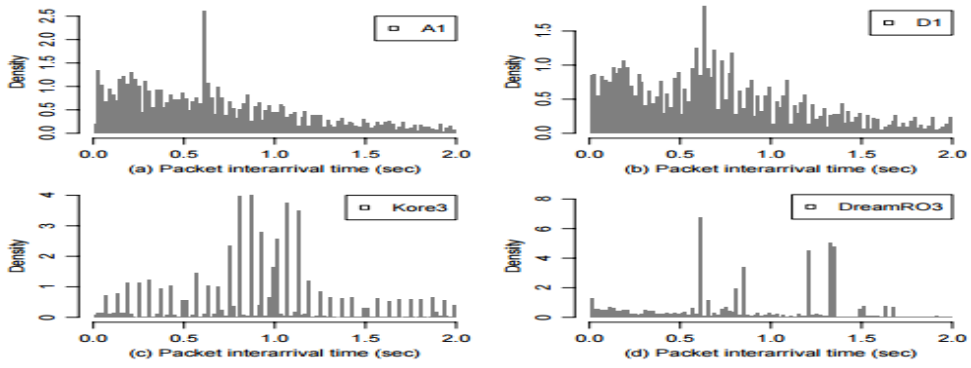

Figure 22 : Histogrammes des horaires de paquets inter-arrivés pour les humains et les bots [15]

Pour juger le degré du hasard des horaires de paquets inter-arrivés, on applique la formule de l'entropie de Shannon H(x), avec x est définit comme variable aléatoire, et pi comme probabilité:

$$H(x)= -\Sigma\ pi\ *log2*pi$$

L'entropie de traces des joueurs humains est plus élevée que celle des traces de bots. On fournit un seuil possible pour le tracé de l'entropie de telle sorte que l'entropie de traces des joueurs est plus élevée que le seuil, tandis que l'entropie de traces des bots est inférieure au seuil. Toutefois, choisir un seuil approprié est difficile, parce qu'on ne peut pas choisir une limite pour l'entropie calculée. En outre, si on calcule l'entropie avec des segments plus larges, l'entropie entre le bot et le lecteur traces sera moins reconnaissable. En effet, avec plus de paquets, et donc plus de temps de paquets inter-arrivés, il y a plus de chance de l'aléatoire dans les temps de paquets inter-arrivés de traces de bot. Pour ces raisons, on n'utilise pas l'entropie du temps de paquets inter-arrivés pour distinguer entre les bots et les joueurs humains.

Si on considère le facteur de temps, on prend successivement des horaires de paquets inter-arrivés dans la même trace comme une série temporelle. On constate que dans certaines traces de bots, des paquets inter-arrivés se produisent périodiquement dans un sens statistique. Prenons l'exemple de *DreamRO*, on envoie 100 paquets inter-arrivée. Sur un ou deux grands écarts de paquet d'environ 2.5 secondes, se produisent environ 10 paquets. Ce modèle régulier est une conséquence du comportement de bot pour certaines tâches, par exemple, dans son processus chasse-monstre, un bot se déplace dans une direction choisie au hasard pour certaines étapes, et répète les étapes jusqu'à ce qu'un monstre soit sur son portée de vue.

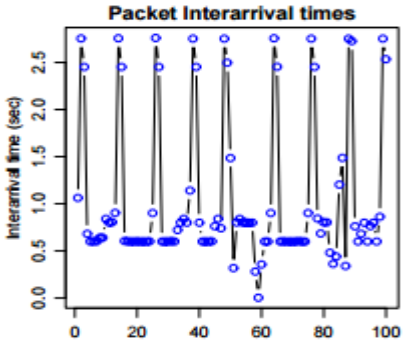

Figure 23 : Horaires de paquets inter-arrivés pour DreamRO [15]

b) SYNCHRONISATION DE COMMANDE :

Pour chaque trace, on calcule les temps de réponse du client pour les paquets de clients qui suivent immédiatement un paquet de serveur. Le temps de réponse de client est la différence de temps

entre une heure de départ de paquets de client et la plus récente heure d'arrivée des paquets de serveur, si aucuns autres paquets de clients n'interviennent. Puis on fait une comparaison entre le temps de réponse de bots et le temps de réponse de joueurs :

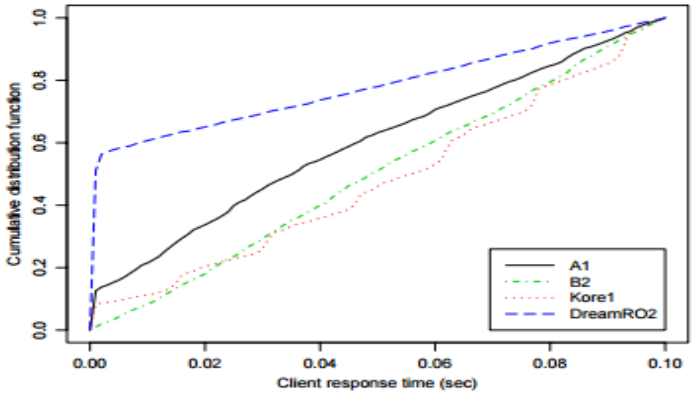

Figure 24: Temps de réponse Client [15]

Comme c'est affiché dans le graphe, sauf pour une augmentation initiale pour A1, les deux traces de joueurs humains, A1 et B2, sont semblables en ce que leurs temps de réponse de moins de 0,1 secondes augmentent en douceur, ce qui signifie qu'ils sont presque uniformément répartis. Alors que la fonction de distribution cumulative pour le bot Kore1 est de type zigzag et les temps de réponse sont regroupés autour de certains intervalles. Parmi toutes les traces de jeu, seulement celles de *DreamRO* possèdent un nombre considérable de temps de réponse courts, ce qu'on appelle des réponses rapides. Par contre, les réponses rapides ne sont pas présentes dans les traces de *Kore*. En effet, il repose toujours sur le serveur d'événements d'arrivée des paquets pour planifier la libération de commandes du client (il garde la régularité).

c) LA SPORADICITE DE TRAFIC

La sporadicité de trafic, à savoir, la variabilité du volume de trafic, ou le nombre de paquets envoyés dans les périodes successivement, est un indicateur de la façon dont le trafic varie au fil du temps. Bien que la sporadicité de trafic est généralement liée à la propriété de mise à l'échelle d'un flux de trafic ou utilisée pour détecter la façon sporadique d'un flux de trafic, on l'a utilisée pour évaluer comment lisser le trafic de bot. L'hypothèse est qu'un bot, en vertu de sa périodicité, doit présenter le trafic lisse par rapport au trafic de joueur.

Plusieurs paramètres sont utilisés pour la sporadicité du trafic :

Coefficient de variation: Le coefficient de variation de moments de paquets inter-arrivée est défini par le rapport de l'écart-type de temps de paquets inter-arrivés au nombre de paquets inter-arrivés.

Après le calcul de coefficient de variation pour des traces de jeux choisies, on obtient :

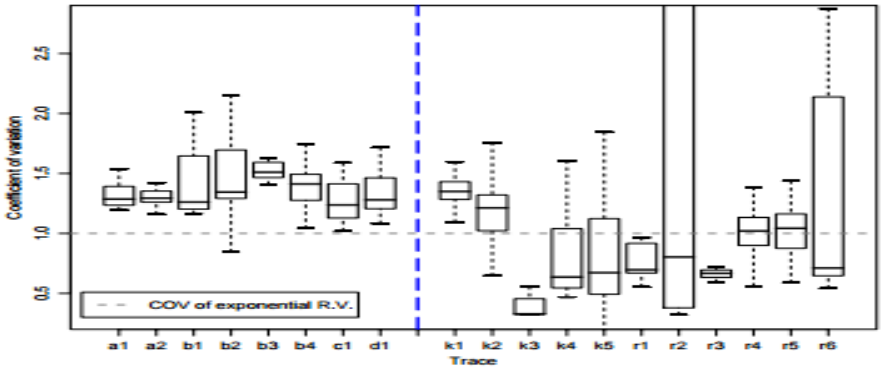

Figure 25 : Coefficient de variation pour les traces de jeu [15]

On constate que les coefficients de variation de la moyenne des traces de joueurs humains (a1, a2, b1, b2, b3, b4, c1, d1) sont tous supérieurs à 1, alors que la plupart des traces de bots (k1, k2, k3, k4, k5, r1, r2, r3, r4, r5, r6) ont des coefficients de variation, inférieurs à 1. Cependant, on ne considère pas que le coefficient de variation soit un indicateur efficace pour différencier les bots et les joueurs humains en raison de l'aléatoire.

Indice de la dispersion de compteurs: on utilise l'indice de dispersion de compteurs pour mesurer la sporadicité du trafic dans différentes échelles temporelles. L'indice de la dispersion des compteurs à l'échelle de temps t est défini comme la variation dans le nombre d'arrivées dans un intervalle de temps t divisé par le nombre moyen d'arrivées en t, qui est :

$$It = Var(Nt) / E(Nt)$$

Où Nt indique le nombre d'arrivées dans un intervalle de temps t.

Après le calcul de l'indice de la dispersion de compteurs pour les traces des jeux, on constate que le taux de paquets de bot *DreamRO* est extrêmement élevé. En effet, il envoie 8 paquets en 0,1 secondes et 15 paquets en 1 seconde, tandis qu'un joueur humain envoie un maximum de 3 paquets en 0,1 secondes et 8 paquets en 1 seconde.

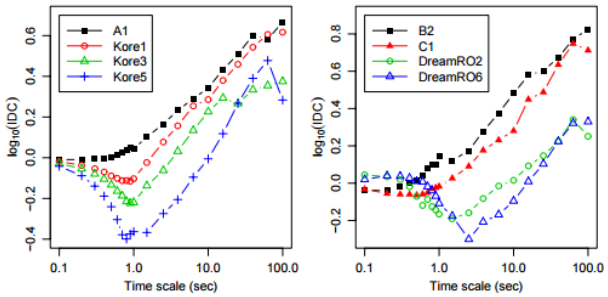

Figure 26: Indice de dispersion des compteurs [15]

Le trafic de bot est plus lisse que le trafic de joueurs, mais il est difficile de définir un seuil pour l'amplitude de sporadicité.

d) LA SENSIBILITE AUX CONDITIONS DU RESEAU

La sensibilité aux conditions de réseau consiste, aux réactions humaines subconscientes à des conditions de réseau intégrées dans des traces de trafic.

Bien que l'allure de jeu est construite par les clients de jeu, elle est inévitablement affectée par les conditions de réseau que les dernières informations sur les autres personnages et l'environnement est acheminé par les paquets de serveur. En bref, le temps de transit des paquets de serveur influe sur le rythme du jeu à jouer et donc le rythme d'un joueur.

Pour évaluer comment le retard de réseau affecte le rythme d'un joueur, on calcule les échantillons de périodes aller-retour ainsi que le taux moyen de paquets au sein de la seconde suivante après chaque échantillon. On constate que le rythme de jeu est lent car les paquets de serveur sont fortement retardés. Par conséquence, les joueurs humains ralentissent inconsciemment leurs actions de clavier et de souris. Par contre, ce phénomène n'apparait pas pour les bots *kore* et *DreamRO*. Comme les bots ont leur propre système de stimulation (certaines fréquences par minute), ils ne sont pas rythmés par les paquets de serveur comme les joueurs humains. En effet, pour un paquet de serveur qui arrive en retard, les bots envoient plusieurs commandes qui sont accumulés avant l'arrivée de ce paquet de serveur.

Dans cette méthode, on a utilisé l'analyse de trafic pour identifier les bots dans les jeux vidéo multi-joueurs en ligne. En particulier, les chercheurs ont appliqué cette méthode sur les traces de jeu *Ragnarok Online*. Ils ont montré que le trafic correspondant à des bots et à des joueurs humains se distingue à plusieurs égards, comme la régularité et les temps de réponse du client, la tendance et l'ampleur de la sporadicité du trafic dans différentes échelles temporelles, et de la sensibilité aux conditions du réseau.

3.4.6 DETECTION DE BOTS EN UTILISANT LES TESTS CONTINUS INTEGRES NON INTERACTIFS

Suite à l'augmentation de l'utilisation des bots dans les jeux vidéo en ligne, les concepteurs de jeux vidéo ont perdu des milliers de dollars puisqu'il ya beaucoup de joueurs déçus qui ont cessé de participer aux jeux. Aussi, les concepteurs de jeux ont perdu des revenus dans les ressources utilisées pour prévenir les différentes formes de tricherie, y compris l'utilisation de bots.

Les chercheurs ont essayé de trouver une solution pour les bots jouant dans les jeux vidéo en ligne, en particulier les jeux avec les résultats financiers réels tels que le *Poker*. Ça consiste à développer des procédures de levée d'ambiguïtés implicites entre homme-machine et développer des tests de intégrés dans le jeu et qui sont continus et non-interactifs. En particulier, Ils ont développé des tests pour les environnements de jeu dans lequel la nature de distraction des essais typiques est particulièrement préjudiciable. Ils ont appliqué ces techniques sur le jeu de *Poker.*

La nouvelle approche proposée par les chercheurs consiste à lire la carte réalisée par le joueur pendant le match. L'identification de la carte elle-même devient un test qui distingue les bots de joueurs humains légitimes.

Voilà un exemple de test intégré qui révèle que la carte est un roi de cœur, s'il est correctement résolu.

Figure 27: Test intégré dans le jeu des cartes [16]

Le test non interactif intégré fonctionne comme suit:

1- L'ordinateur génère une instance de test avec une solution correspondant à la carte distribuée.

2-essai est représenté par l'humain / bot;

3- humain / bot tente de résoudre le test;

Si le test est résolu correctement, les décisions futures de l'humain / bot peuvent être intelligents, comme l'humain / bot a toutes les informations nécessaires pour prendre des décisions. Sinon, l'information sur laquelle les décisions sont fondées est défectueuse, et, partant, les décisions ne sont pas optimales et l'humain / bot essentiellement agit inintelligemment.

Cette méthodologie a plusieurs avantages. Tout d'abord, il ya moins de distraction de la tâche à accomplir, comme le test est intégré dans l'application et ne constitue pas une tâche distincte nécessitant un être humain d'accomplir une activité sans rapport tout en prenant une pause forcée de

l'application principale. Ensuite, le bot ne sait pas si le test a été réussi ou échoué, et donc ne peut pas apprendre de ses erreurs pour améliorer ses performances pour les présentations futures du test. Aussi, les bots ne sont pas seulement empêchés d'obtenir une ressource, mais sont effectivement punis pour avoir tenté d'accéder à une ressource, ce qui rend moins probable que les futures tentatives d'obtenir la ressource suivront. Puis, un joueur humain qui se rend compte qu'il ou elle a résolu le test de manière incorrecte, peut résoudre correctement à un point avenir. Finalement, depuis qu'aucune solution à un test est disponible, l'externalisation que le test deviendra impossible, que n'importe quelle réponse générée par le solveur sera acceptée mais ne serait vérifiable comme précise.

Voilà un exemple de salle de poker avec trois cartes du flop codées pour empêcher les bots de jouer ou de collecter d'informations sur le jeu. Ces tests assurent la participation continue de joueur humain pendant la durée du jeu.

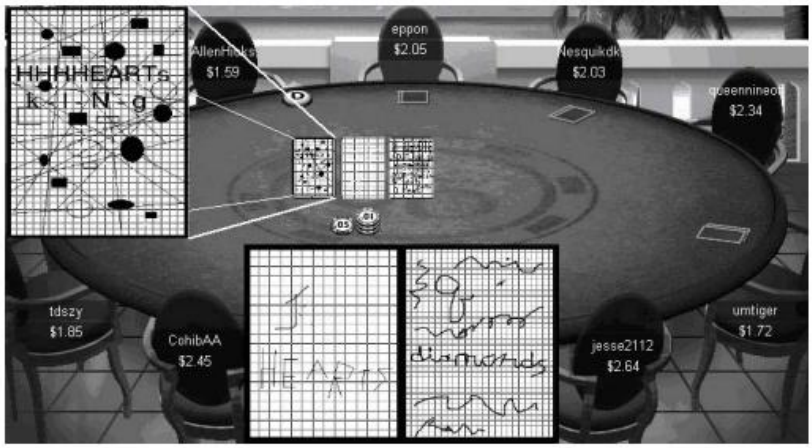

Figure 28: Cartes de poker avec des tests intégrés [16]

Suite à cette expérience, on peut conclure qu'un test non interactif intégré peut être utilisé pour empêcher les bots de participer illégalement dans les jeux en ligne.

3.5 PREVENTION CONTRE LES BOTS

Plusieurs techniques peuvent être utilisées pour la détection des bots dans les jeux vidéo. Tout d'abord, la technique de **présence humaine** qui impose la présence humaine dans un jeu si certaine entrée d'un être humain est appelée à jouer le jeu. L'exigence est que seulement un humain contribue une partie de l'interaction avec le jeu, mais pas nécessairement la totalité. En particulier, l'être humain peut être assisté par un bot qui contrôle le reste de l'interaction avec le jeu. Une autre technique utilisée qui exige que toutes les interactions avec le logiciel de jeu viennent d'un humain, sans aucune intervention d'un bot. Finalement, la technique de **modèle contradictoire**. En effet, les joueurs peuvent réussir dans l'ingénierie inverse ou d'interférer avec le code en cours d'exécution sur leurs machines locales, mais le code local est utilisé seulement pour échanger des données avec les serveurs centraux.

Un test CAPTCHA (Completely Automated Public Turing Test to Tell Computers and Humans Apart) est un programme qui peut générer des tests que la plupart des êtres humains peuvent passer, mais que les programmes informatiques actuels ne peuvent pas passer. Ces tests ont une nécessité heuristique. En effet, il n'y a pas un moyen de prouver qu'un programme d'ordinateur ne peut pas passer un test que l'homme peut passer. Ils exploitent les écarts entre les capacités des humains et celle des ordinateurs mais il n'y a aucune garantie que ces lacunes ne vont éventuellement disparaître.

L'algorithme classique de CAPTCHA est:

1- L'ordinateur génère une instance de test

2- Un essai est représenté à l'humain / bot

3- L'humain/le bot tente de résoudre l'instance de l'essai

4-Les rapports humains / bot prétendue solution à l'ordinateur

5- L'ordinateur évalue la solution présentée

6-Selon le résultat de l'évaluation à l'humain / bot l'ordinateur autorise ou bloque l'accès à une ressource basée sur le résultat.

Le test CAPTCHA est présenté par le schéma ci-dessous :

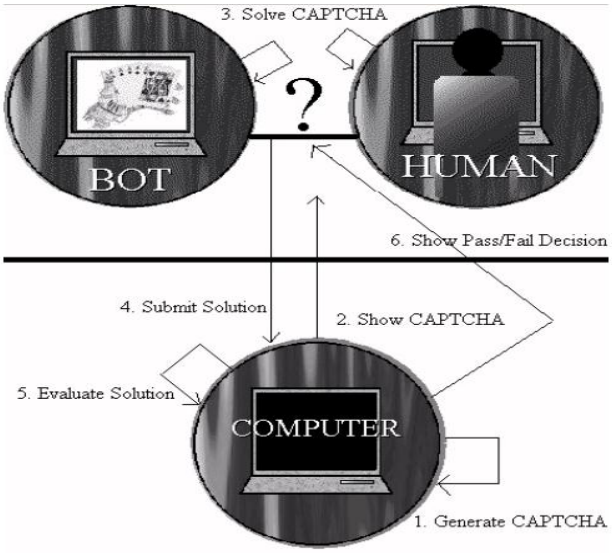

Figure 29: Test de CAPTCHA [17]

Les tests CAPTCHA les plus utilisés reposent actuellement sur la capacité des humains de reconnaître du texte ou des images déformées de manière aléatoire. Aussi, ils peuvent être utilisés lors d'un match pour détecter une la présence humaine. Un large éventail de stratégies d'essai est possible. Leur avantage principal est qu'ils sont extrêmement souples et peuvent être utilisés à peu près avec tous les jeux. Cependant, les tests CAPTCHA souffrent de plusieurs limitations. Tout d'abord, les tests CAPTCHA qui perturbent. En effet, **i**ls attirent l'attention du lecteur à l'écart du jeu. L'interruption est

seulement momentanée, mais il se casse la suspension de l'incrédulité qui est si important pour les jeux. Les tests CAPTCHA peuvent aussi nuire à l'allure d'un jeu. Ils causent des demandes fréquentes pour résoudre les tests CAPTCHA ce qui serait intolérable pour les joueurs humains, car le rythme et le sens de l'écoulement est essentiel à la jouissance d'un jeu. Aussi, les tests CAPTCHA peuvent être externalisés. En effet, Un CAPTCHA doit être résolu par un être humain, mais que l'homme ne doit pas être la personne qui joue la partie.

CHAPITRE 4

APPROCHES PROPOSEES POUR LA DETECTION DES BOTS DANS LES JEUX VIDEO MULTI-JOUEURS EN LIGNE BASEE SUR LE FORAGE DE DONNEES

4.1 INTRODUCTION

Dans le chapitre précédent, nous avons cité les différentes méthodes qui existent pour la détection de bots dans les jeux vidéo en ligne. Les résultats obtenus au cours des recherches antérieures permettent de faire la différence entre un joueur humain et un bot en se basant à chaque fois sur un seul critère. En effet, les chercheurs ont choisi d'approfondir leurs recherches sur certains critères en négligeant d'autres. Comme but de notre projet, nous avons choisi de classifier les traces de jeu et faire la différence entre les joueurs humains et les bots en se basant sur plusieurs critères.

Dans ce chapitre, nous allons expliquer le travail réalisé qui consiste à construire un modèle d'agent (un modèle d'agent est un profil avec des critères bien précis) pour un joueur humain et un autre modèle d'agent pour les bots, en se basant sur plusieurs critères et plusieurs expériences. Puis selon ces modèles, nous avons classifié les traces qu'on possède en traces humaines et traces de bots.

Ce travail a pour objectif de détecter les bots dans les jeux vidéo en ligne multi-joueurs en appliquant les techniques de forage de données. C'est une solution générale qui pourra être appliquée à n'importe quel jeu vidéo en ligne où le joueur contrôle le déplacement de son avatar.

En réalisant des expériences sur des traces humaines et des traces de bots, nous avons trouvé que les deux types d'agents (humain et bot), ont des comportements complètements différents. Ce qui nous a aidés à construire deux modèles de comportements différents pour chacun de ces agents.

De plus, nous avons fait l'analyse de déplacements des joueurs humains et des bots afin de déterminer s'ils se déplacent de même manière ou non (directions, avancements, reculement...).

4.2 DESCRIPTION DE DONNEES

4.2.1 QUAKE2

Quake2 est un jeu vidéo en ligne multi-joueurs. Il s'agit d'un jeu de tir à la première personne (First Person Shooter), qui a été développé par *id Software* et publié par *Activision* en 1997[12]. Comme tous les jeux de tir à la première personne, le joueur adopte le rôle d'un caractère particulier et tire ses ennemis via l'interface utilisateur.

Figure 30: Imprime écran de Quake2 [www.deviantart.com]

Les jeux de morts (Games of Death), dans lesquels chaque joueur tente à tuer autant d'autres joueurs que possible, sont très populaire. *Quake2* a été nommé comme le meilleur jeu en 1998, et s'en vendu plus d'un million de copies [12]. Une des raisons de popularité de jeu est qu'il est facile à utiliser, et un grand nombre de cartes, des modèles de joueurs, des textures et des effets sonores sont disponibles sur internet.

4.2.2 LES TRACES HUMAINES

Quake2 possède une fonction d'enregistrement, qui permet de suivre chaque action et mouvement, ainsi que le statut de chaque personnage et un objet tout au long de jeu. Avec une trace enregistrée, on peut reconstruire un jeu et l'examiner de toute position. Les joueurs utilisent souvent cette fonction pour évaluer leurs stratégies de performance et de combat. De plus, les joueurs expérimentés sont invités à publier leurs traces de jeu comme matériel didactique pour les joueurs débutants et ainsi faire une réputation dans la communauté. Les traces humines sur lesquelles nous

avons fait nos expériences sont téléchargées de sites de jeu : *GotFrag Quake*[1], *Planet Quake*[2], *Demo Squad*[3] et *Revilla Quake Site*[4]. [5]

4.2.3 LES TRACES DE BOTS

Il y existe plusieurs bots de jeux pour Quake2. Au cours de ce projet nous avons travaillé avec les trois bots les plus populaires pour *Quake 2*. Ces bots sont : *CR BOT*, *ERASER BOT* et *ICE BOT*. Chaque trace de bot est d'une durée d'environ 20h.

Les traces de bots que nous avons utilisées, ont été récoltées au sein de centre de sécurité de l'information de Taiwan (*Taiwan Information Security Center* (TWISC)) par Chen ainsi que ses collègues [4].

Voilà quelques échantillons d'une trace de Crbot :

Tableau 3 : Échantillons de la trace de *Crbot*

Time	X	Y	Z	Ox	oy	Oz
20	1328.0	932.0	1208.0	0.0	0.0	0.0
21	1328.0	924.0	1208.0	0.0	-8.0	0.0
22	1343.875	928.0	1224.25	15.875	4.0	16.25
23	1359.875	928.0	1240.5	16.0	0.0	16.25
24	1375.875	920.0	1256.75	16.0	-8.0	16.25
25	1391.875	920.0	1273.0	16.0	0.0	16.25
26	1407.75	920.0	1289.25	15.875	0.0	16.25
27	1423.75	920.0	1305.625	16.0	0.0	16.375
28	1439.75	920.0	1321.875	16.0	0.0	16.25
29	1455.75	920.0	1338.125	16.0	0.0	16.25

4.3 ANALYSE DE DONNEES

Au cours de cette phase nous avons commencé par l'analyse des traces recueillies (traces humaines et traces de bots). En effet, nous avons calculé pour chaque trace la distance moyenne (la distance que l'agent a faite pendant une certaine durée), la vitesse moyenne ainsi que la période où l'agent est resté inactif. Ensuite, nous avons dressé les courbes représentatives de variation de vitesses

[1] http://www.esreality.com/post/1731494/re-gotfrag-quake-classic/
[2] http://planetquake.gamespy.com/
[3] http://q2scene.net/ds/
[4] http://www.revilla.nildram.co.uk/demos-full.htm

pour les traces de chaque type d'agent ainsi qu'un tableau comparatif entre les différents agents. Enfin, nous avons classifié les traces en se basant sur la vitesse.

4.3.1 LA VITESSE

La vitesse totale ou la vitesse moyenne d'une trace est calculé en fonction du coordonnées (x, y, z) des positions des agents. C'est est le rapport de distance parcourue divisée par le temps (ou la période) où l'agent a fait son déplacement.

C'est-à-dire : **Vitesse moyenne = distance parcourue/temps de parcours**

Après avoir calculé la durée totale pour chacune des traces, nous avons constaté que la durée d'une trace est presque la même pour chacun de types de traces (c'est-à-dire la durée d'une traces des bots est presque la même pour toutes les traces de bots et la durée d'une trace humaine est presque la même pour toutes les traces humaines). Ce qui varie ici c'est la distance. C'est pour cela nous avons calculé la distance puis la vitesse pour chacune de traces. Ensuite, nous avons calculé la période où l'agent n'a pas bougé (période de repos).

Voilà un tableau comparatif entre les différents types d'agents (*CRbot, Eraser, Ice* et joueur humain) :

Tableau 4 : Tableau comparatif entre les traces

Agent	**Humain**		***Crbot***		***Eraser***		***Ice***	
	Minimum	Maximum	Minimum	Maximum	Minimum	Maximum	Minimum	Maximum
Distance	162.75	472361.81	1226193	1829598	1893595	2244271	1676481	1817070
Vitesse	0.0017	0.44	17.04	25.43	25.74	31.23	23.41	25.29
Période d'inactivité	2h et 8 min	1 jour et 3 h	54.5 min	2 h et 2 min	42.2 min	2h et 10 min	5h et 42 min	7h

D'après ce tableau on constate que le joueur humain est le moins rapide. En effet, la distance qu'il fait ne dépasse pas 473.000 mm dans une longue période. En plus, sa période d'inactivité (la période de repos) est de minimum 2h et 8 min allant jusqu'à 27h. Par contre, pour les bots, on voit qu'ils sont très rapides. Par exemple : la vitesse d'*ERASER* peut aller jusqu'à 31.23 mm/s, et leurs périodes d'inactivité sont petites par rapport à celles de joueurs humains.

Ces différences sont causées par le fait qu'un bot n'est pas un humain et un humain n'est pas un bot. Un bot est un programme informatique qui peut rouler des heures et des jours sans cesse. Alors qu'un joueur humain peut prendre une pause, manger, dormir… De plus, un joueur humain ne peut pas être aussi rapide qu'un bot puisque ce sont les programmeurs de bots qui contrôlent la vitesse de bots. Ainsi, un joueur humain peut se fatiguer ou se démotiver ce qui diminue sa vitesse de plus en plus alors que ce n'est pas le cas pour un bot.

4.3.2 COURBES DE VITESSES

Les courbes de vitesses ont été dressées en fonction de vitesses moyennes de traces. Ici l'axe des X représente le numéro de la trace et l'axe des Y représente la vitesse moyenne de la trace.

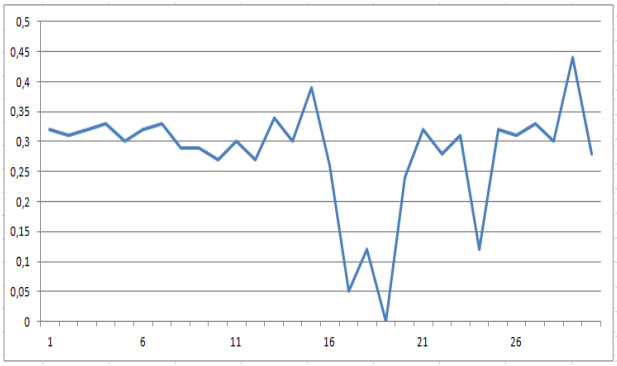

Figure 31 : Courbe de vitesse de joueurs humains

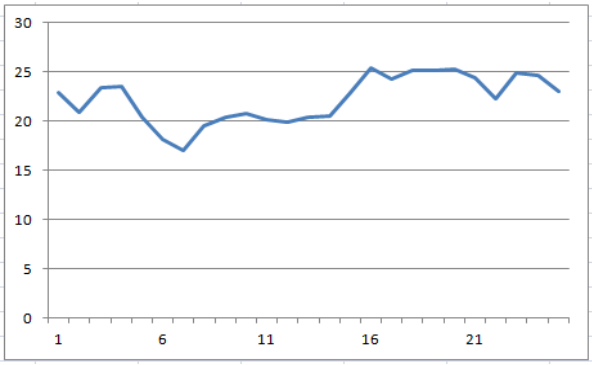

Figure 32 : Courbe de vitesses de Crbot

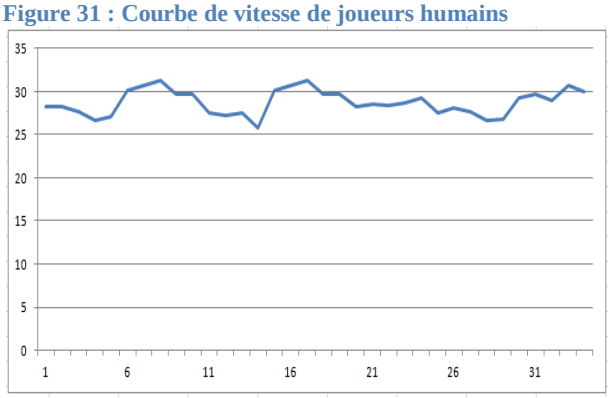

Figure 33 : Courbe de vitesses d'Eraser

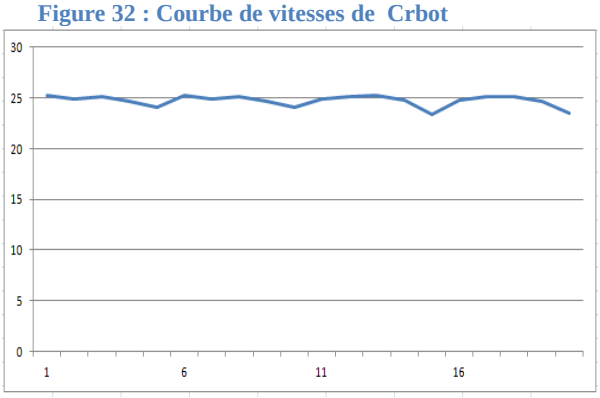

Figure 34 : Courbe de vitesses d'Ice

Comme mentionné dans le tableau ci-dessus, et comme le montre la figure 31, la vitesse maximale pour les joueurs humains ne dépasse pas 0.44mm/s ce qui est logique pour un joueur humain qui déplace la souris et se sert de la console.

Pour les courbes de vitesses de bots, on voit bien sur les figure 32,33 et 34 que les vitesses des bots *Crbot*, *Eraser* et *Ice* sont beaucoup plus élevées par rapport à celles des joueurs humains.

Ceci est dû à la capacité des bots de réaliser des grandes distances dans une durée limitée. En effet, le bot est un programme informatique qui possède des instructions bien précises, des formules mathématiques bien calculées. Un bot n'a pas besoin du temps pour réfléchir et prendre une décision

concernant sa prochaine réaction ou sa prochaine direction. Il fonctionne en utilisant les techniques de l'intelligence artificielle. En fait, les programmeurs de bots connaissent très bien *Quake2*, donc ils ont mis des solutions pour tous les problèmes qu'un bot pourrait avoir. En plus le bot connait très bien les obstacles qu'il peut les avoir en jouant et la façon de réagir face à ces problèmes. Ainsi, il connait bien ce qu'il faut faire au niveau de chaque position de la carte de jeu.

Comme il ne connait pas à l'avance les surprises qu'il pourrait avoir en jouant, le joueur humain doit à chaque fois, regarder, réfléchir puis agir face à un problème ou un obstacle ce qui lui perd du temps et ce qui augmente sa période d'inactivité. De plus un joueur humain déplace la souris et utilise le clavier pour déplacer son avatar, pour tuer ses ennemis, pour se cacher, ce que lui prend du temps aussi.

Tous cela explique la vitesse élevée de bots qui varie entre 17 mm/s et 31 mm/s pour les bots et qui ne dépasse pas 0.44mm/s pour les joueurs humains. En moyenne, un bot est 55 fois beaucoup plus rapide qu'un joueur humain.

Comme premier résultat, nous pouvons déduire que la vitesse moyenne de trace est un critère avec lequel nous pouvons faire la différence entre un joueur humain et un bot ce qui nous aide à détecter les bots en utilisant les approches de forage de données.

4.4 VITESSES SUR DES INTERVALLES DE 200 S

Au début de travail, nous avons commencé par travailler sur les traces entières. Afin d'approfondir nos recherches, nous avons divisé chaque traces en des intervalles de 200 s, pour pouvoir mieux analyser leurs vitesses moyennes et les changements de vitesse que les agents ont faits. Nous avons travaillé sur une trace de *Crbot*, une trace de *Eraser*, une trace d'*Ice* et une trace humaine. Nous avons découpé chacune de ces 4 traces en des intervalles de 200 s, puis nous avons calculé la vitesse sur ces intervalles.

4.4.1 COURBES DE VITESSES

Les courbes de vitesses obtenues :

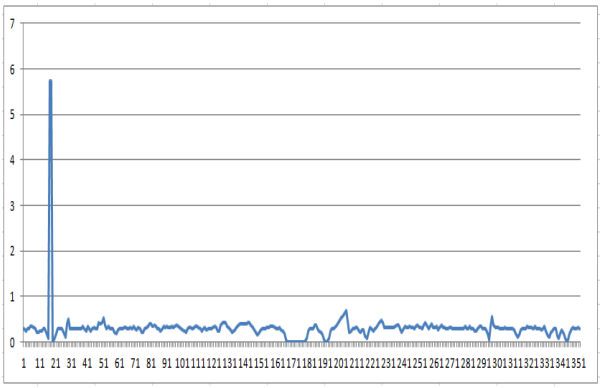

Figure 35 : Vitesses sur des intervalles de 200s pour humain

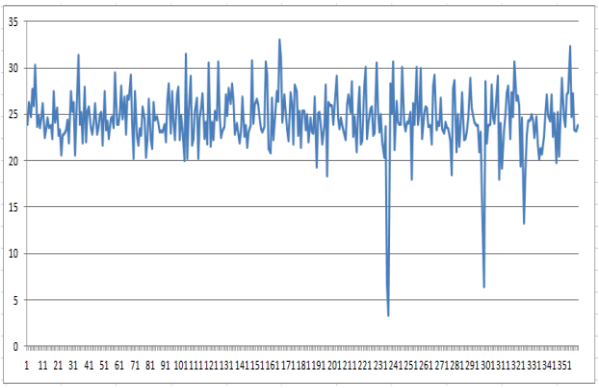

Figure 36 : Vitesses sur des intervalles de 200s pour Crbot

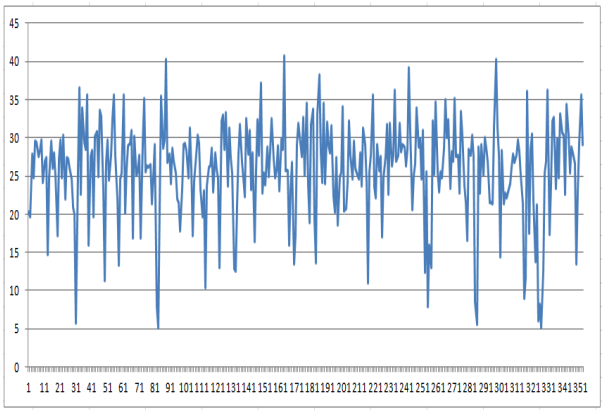

Figure 37 : Vitesses sur des intervalles de 200s pour Eraser

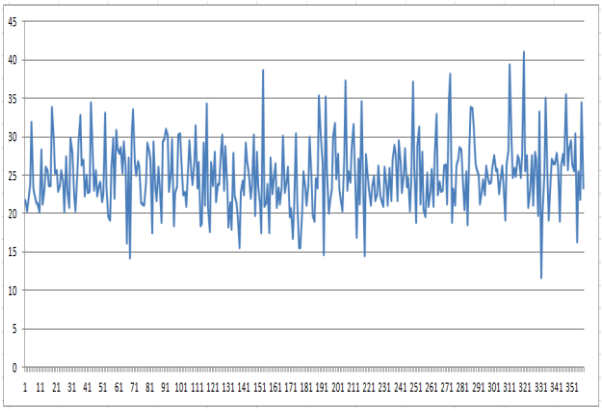

Figure 38 : Vitesses sur des intervalles de 200s pour Ice

Sur ces courbes nous voyons très bien la différence de vitesses sur les intervalles de 200 s entre le joueur humain et les bots. En effet, la vitesse de joueur humain est généralement en dessous de 1 mm/s sauf qu'il y'a un pic qui dépasse 5 mm/s. Par contre, pour les 3 bots, nous voyons que les vitesses sont généralement entre 20 et 30 mm/s avec des pics qui parfois dépassent 40 mm/s ou des diminutions allant jusqu'à moins de 5 mm/s. Ceci est normal car les bots sont des programmes informatiques très rapides.

4.4.2 CLASSIFICATION DES VITESSES DE 200 S

Les valeurs de vitesse calculées sur des intervalles de 200 s de 4 traces (*Crbot*, *Eraser*, *Ice*, et joueur humain) ont été stockées dans des fichiers CSV qui ont été convertis en fichiers ARFF.

Un fichier CSV est un fichier texte, sous lequel on stocke nos données. Chaque ligne de fichier CSV est une ligne de tableau où les valeurs de chaque colonne sont séparées par un critère de séparation (en général une virgule ou un point-virgule).

Dans le fichier CSV, toutes les lignes contiennent obligatoirement le même nombre de valeur et le même nombre de caractères de séparation.

La syntaxe de fichier CSV:

```
[NomColonne1, NomColonne2,….., NomColonneN]
ValeurColonne1,ValeurColonne2,…,ValeurColonneN
ValeurColonne1,ValeurColonne2,…,ValeurColonneN
ValeurColonne1,ValeurColonne2,…,ValeurColonneN
….
```

Exemple :

```
[Vitesse,Distance,Agent]
23,924,Bot
26,266,Bot
24,774,Bot
0.2,394,Humain
0.1,312,Humain
```

Le fichier *ARFF* (Attribute-Relation File Format) est un fichier texte *ASCII* (American Standard Code for Information Interchange) qui contient une liste d'instances partageant un ensemble d'attributs. Les fichiers *ARFF* ont été développés au sein de département des sciences informatiques de l'université de *Waikato* [18] pour une utilisation avec le logiciel d'apprentissage automatique *weka*.

Un fichier *ARFF* est constitué de deux parties : l'entête (Header) et les données (Data information). L'entête contient le nom de la relation c'est-à-dire le nom de base de données qu'on va classifier, la liste des attributs et leurs types (exemple :Numeric, string), . La partie de données décrit les instances.

La syntaxe de fichier ARFF :

```
@RELATION  nomRelation
@ATTRIBUTE   nomAttribut1
@ATTRIBUTE  nom Attribut2
@ATTRIBUTE  nomAttribut3
…
@ATTRIBUTE class {‘nomClass1’ , ‘nomClass2’ , …, ‘nomClassN’}

@DATA
ValeurAttribut1, ValeurAttribut2, ValeurAttribut3,…, ‘nomClass’
ValeurAttribut1, ValeurAttribut2, ValeurAttribut3,…, ‘nomClass’
ValeurAttribut1, ValeurAttribut2, ValeurAttribut3,…, ‘nomClass’
ValeurAttribut1, ValeurAttribut2, ValeurAttribut3,…, ‘nomClass’
ValeurAttribut1, ValeurAttribut2, ValeurAttribut3,…, ‘nomClass’
```

Exemple de fichier ARFF :

```
@RELATION  Vitesse200s
@ATTRIBUTE   Vitesse
@ATTRIBUTE  Distance
@ATTRIBUTE class {‘Bot’ , ‘Humain’}

@DATA
23,924, ’Bot’
26,266, ‘Bot’
24,774, ‘Bot’
0.2,394, ‘Humain’
0.1,312, ‘Humain’
…..
```

Ces fichiers ARFF constituent nos base de données qu’on va les classer avec *weka* (logiciel de forage de données).

La figure suivante montre l’importation de nos bases de données sous weka :

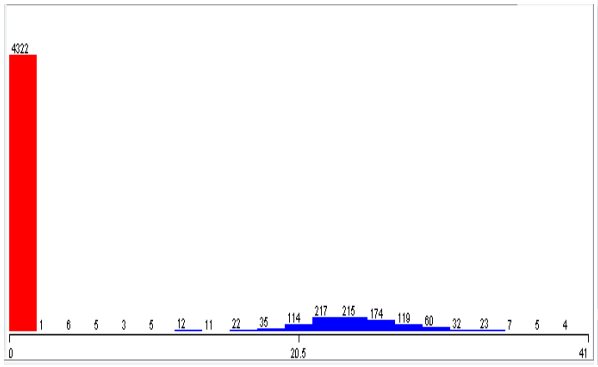

Figure 39: Base de données sous weka

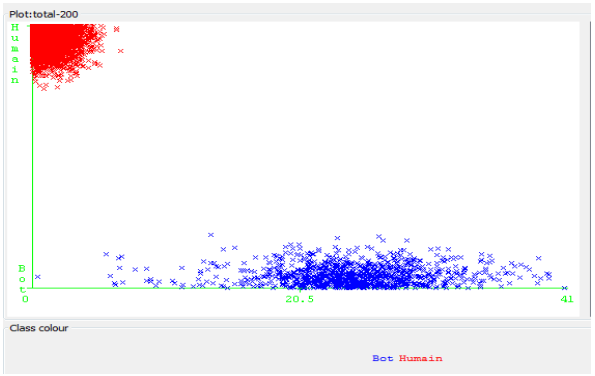

Figure 40: Nuage de points

L'axe X montre que les valeurs de vitesse sur des intervalles de 200 s varient entre 0 et 41 mm/s.

La figure 39 montre que nous avons deux classes et la figure 40 montre le nuage de points. En effet, les points (instances) de classe Bot sont représentés en bleu et les points de la classe Humain sont représentés en rouge.

Afin de mieux connaitre ces classes et leurs caractéristiques, une classification supervisée et une classification non supervisée, ont été faites.

a) L'ALGORITHME *J48*

J48 est un algorithme implémenté sous *weka* qui nous permet d'établir une classification supervisée de nos données et de construire un arbre de décision. C'est l'implémentation java de l'algorithme *C4.5* qui est à son tour un algorithme de classification supervisée, implémenté sous *weka*. Il crée un arbre binaire. *J48* permet de classer les données soit par des arbres de décisions, soit par des règles générées par ceux-ci.

Dans notre cas, nous avons une base de données de taille 5392 (5392 instances), où chaque instance représente la vitesse de l'agent sur un intervalle de 200s. En utilisant un cross validation égale à 10% (le cross validation est l'ensemble d'apprentissage et de la construction d'arbre de décision), nous avons fait la classification de nos données en appliquant l'algorithme *J48* :

```
Number of Leaves  :     2
Size of the tree :      3

Time taken to build model: 0.09 seconds

=== Stratified cross-validation ===
=== Summary ===
Correctly Classified Instances        5389               99.9444
%
Incorrectly Classified Instances      3                  0.0556 %
Kappa statistic                       0.9983

Total Number of Instances             5392
```

Figure 41: Classification supervisée des vitesses sur des intervalles de 200 s avec J48

Number of Leaves: est nombre de feuilles c'est-à-dire les nœuds finaux qui sont nos classes.

Size of the tree : est le nombre de tous les nœuds qui constituent l'arbre.

Correctly Classified Instances: est le taux de classification (les instances bien classées)

Incorrectly Classified Instances: est le taux d'erreur (les instances mal classées)

Kappa statistic: est le degré de concordance de deux ou plusieurs juges (l'accord/ désaccord entre deux juges) et cette valeur se lit à la matrice de confusion.

Dans notre cas la matrice de confusion est :

```
=== Confusion Matrix ===

   a    b   <-- classified as
1068    0 |    a = Bot
   3 4321 |    b = Humain
```

Suite à une classification supervisée de nos données en utilisant l'algorithme *J48*, un arbre de décision a été construit en 0.09 s. Cet arbre est construit de deux classes : Bot et Humain. Le seuil de classification est 1. En effet, si la vitesse d'un intervalle de 200s est <=1, il s'agit d'un joueur humain qui joue sinon si la vitesse d'un intervalle de 200s > 1 alors s'agit d'un bot.

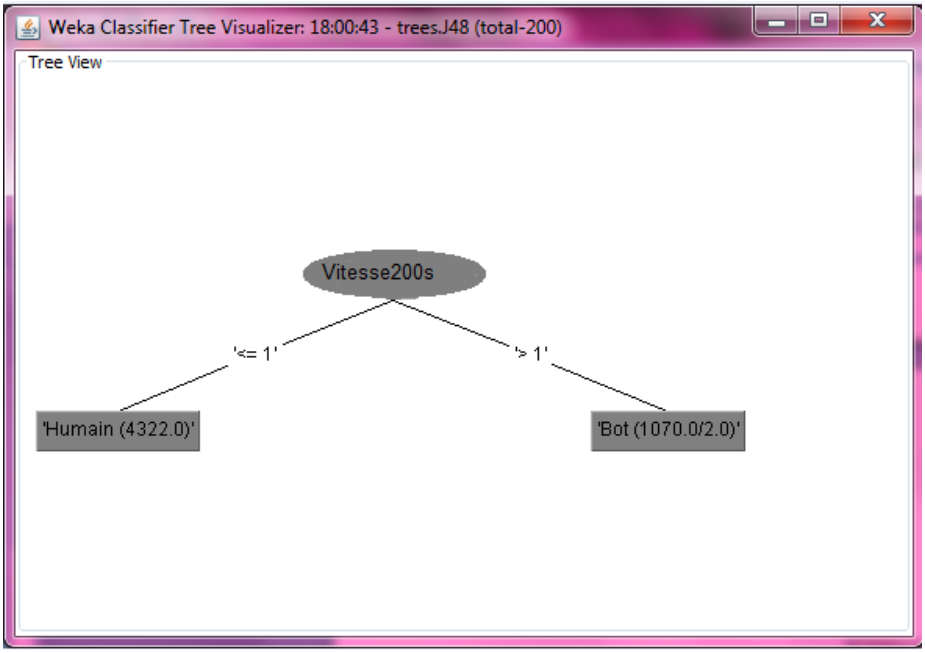

Figure 42: Arbre de décision avec J48

Les instances ont été classées à 99.9444 % correctement.

Afin de valider nos résultats de classification supervisée faite avec l'algorithme *J48*, une autre classification non supervisée a été établie, en utilisant l'algorithme *SimpleKMeans.*

b) L'ALGORITHME SIMPLE K-MEANS

L'algorithme *K-Means*, un des algorithmes de *clustering* les plus utilisés, est classifié comme un algorithme non hiérarchique. Il analyse la base de données numériques de taille N. Puis il devise cette base en K clusters (groupes).Les membres de chaque cluster sont similaires entre eux et dissimilaire avec les membres des autres clusters.

La procédure de l'algorithme *K-Means* suit une manière simple et facile pour classifier un ensemble de données en des clusters fixés à priori.

Algorithme des *K-Means* :

Nécessite 2 paramètres : le jeu de données X, le nombre de groupe à constituer k ∈ N

Prendre K centres arbitraires x j ∈ D

Répéter

Pour j ∈ {1,…,K} faire

Gj ← Ø

FinPour

Pout i ∈ {1,….,N} faire

j*← arg min d(xi, cj)

Gj*← Gj* ∪ {xi}

FinPour

Pour j ∈ {1,…,K} faire

cj← centre de gravité de Gj

finPour

Recalculer Iw

Jusqu'au Iw < seuil

Figure 43 : Algorithme de K-Means [18]

L'objectif de l'application de l'algorithme de *K-Mean* est de segmenter les données en k-groupes, k étant fixé à l'avance [18].

On part de K données synthétiques. Ces points font partie de l'ensemble de points de l'espace de données D. Ces points sont nommés centres et chaque centre caractérise un groupe. A chaque centre sont associées les données qui lui sont proches : cela crée un groupe autour de chaque centre [18].

Au début, tous les groupes (N) ne possèdent pas de centre de gravité (Gj ← Ø). On calcule le centre de gravité de chacun de ces groupes (j*← arg min d(xi, cj) Gj*← Gj* ∪ {xi}). En effet un centre de gravité G, d'un ensemble de données X, est une donnée synthétique dont chaque attribut est égal à la moyenne de cet attribut dans X [18], soit :

G= (M1,M2,…, Mp)

Où Mi est la moyenne de l'attribut Ai.

Ces K centres de gravité deviennent les nouveaux centres et on recommence tant que les groupes ne sont pas stabilisés. La stabilité signifie qu'il n'y a pas de données qui changent de groupe d'une itération à la suivante. Ou encore l'inertie I (l'inertie est la distance entre les données et les centres de gravité des groupes) ne varie pas substantiellement d'une itération à la suivante. La formule de l'inertie est :

$$\mathbf{I = \sum d^2 (Xi, G)}$$

Où G est le centre de gravité de X, et Xi est la iemme donnée -individu- de l'ensemble de données de X [18].

L'algorithme de *K-Means* converge en un nombre d'itérations.

Nous avons appliqué l'algorithme *SimpleKMeans* sur notre de base de données :

```
KMean
Number of iterations: 4
Within cluster sum of squared errors: 167.21785773986787
Missing values globally replaced with mean/mode
Cluster centroids:
                        Cluster#
Attribute    Full Data        0        1
                  (5363)    (1030)    (4333)
=========================================
23             4.7218    24.5748    0.0025
924          324.465   491.6194   284.7307
Bot           Humain       Bot     Humain
Time taken to build model (full training data) : 0.02 seconds
=== Model and evaluation on training set ===
Clustered Instances
0      1030 (19%)
1      4333 (81%)
```

Figure 44: Classification non supervisée des vitesses sur des intervalles de 200 s avec *SimpleKMeans*

***Within cluster sum of squared errors**: est l'inertie interclasses. C'est-à-dire la distance entre les clusters. La formule de l'inertie interclasses est :

$$\mathbf{Iw= \sum Wi * d^2 (gi, g)}$$

Où gi est le centre de gravité du groupe Gi et g est le centre de gravité de l'ensemble de données X [18].

Le résultat de la classification non supervisée avec l'algorithme de *K-MEANS* est :

Clustered Instances	
0	1030 (19%)
1	4333 (81%)

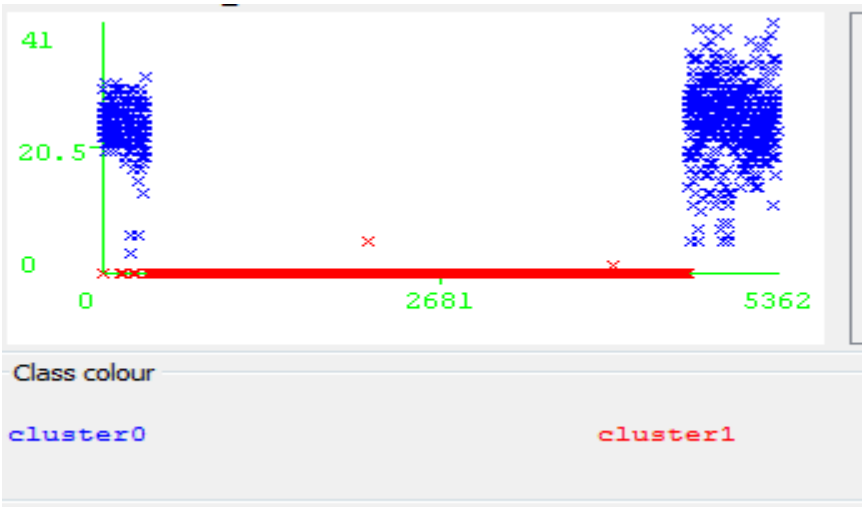

Figure 45: Clusters de des vitesses sur des intervalles de 200 s (Résultat de classification avec l'algorithme *SimpleKMesans*)

Nous avons obtenu deux clusters : Cluster 0 en bleu et cluster 1 en rouge.

19% des instances ont été classées dans le cluster 0 qui a des valeurs à l'alentour de 0. Et le reste des instances (81%) ont été classées dans le cluster1.

Une classification non supervisée avec l'algorithme *SimpleKMeans*, confirme nos résultats de la classification avec l'algorithme *J48* et donne deux clusters : cluster0 et cluster1 comme le montre la figure ci-dessus.

Avec ces résultats, nous pouvons déduire que la vitesse moyenne d'un intervalle de 200s est un critère qui nous aide à détecter les bots en utilisant les approches de forage de données. L'objectif de calculer la vitesse de différents agents sur des intervalles de 200s est de déterminer quand est ce que un joueur humain joue lui-même et quand est ce que il met un bot pour jouer à sa place. Donc d'après ces résultats la vitesse d'un agent sur un intervalle de 200s peut être un indice de détection de bots dans les jeux vidéo en ligne.

4.5 CALCUL DE RATIOS DE VITESSES SUR DES INTERVALLES DE 200 S

4.5.1 COURBES DE RATIOS DE VITESSES

Après avoir calculé la vitesse moyenne des traces ainsi que les vitesses moyennes sur des intervalles de 200 s pour les traces, nous passons à calculer les ratios de vitesses. En effet, le ratio de vitesse est la vitesse d'un intervalle de 200 s divisée par la vitesse moyenne de la trace.

C'est-à-dire **: ratio = vitesse sur l'intervalle de 200s/vitesse moyenne de la trace entière**

Nous avons fait cette expérience sur une trace d'un joueur humain ainsi que les traces de trois bots *Crbot*, *Eraser* et *Ice*. Voilà ce que nous a donné :

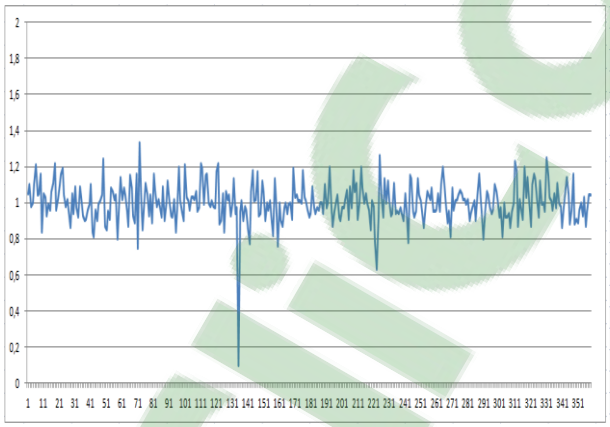

Figure 46: Les ratios pour Crbot

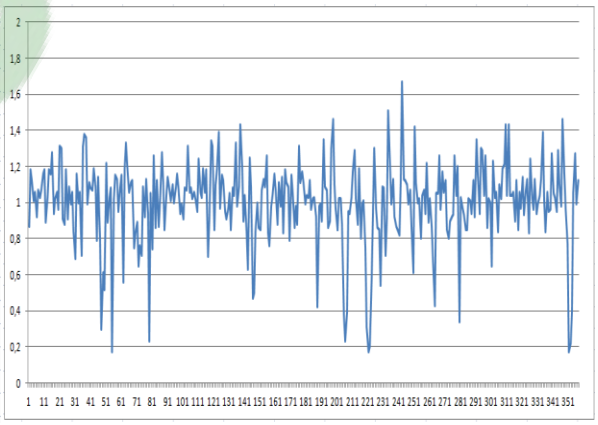

Figure 47: Les ratios pour Eraser

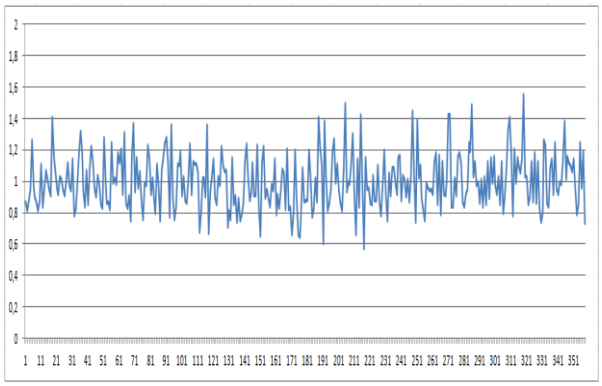

Figure 48 : Les ratios pour Ice

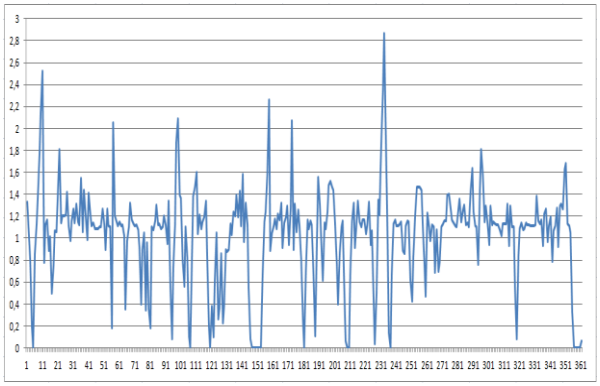

Figure 49 : Les ratios pour le joueur humain

D'après les figures 46, 47, 48 et 49 nous remarquons que le ratio de vitesse varie beaucoup pour le joueur humain (fig. 49) que pour les bots. De plus le ratio pour le joueur humain est plus élevé allant jusqu'à 2.8. Il fait des grands pics.

Ceci est dû aux changements de vitesses pour les humains. Il a des zéros (ce sont des périodes de repos), ce qui n'est pas le cas pour les bots car ce sont des programmes. Ils n'ont pas de périodes de repos. En plus les valeurs de leurs ratios varient généralement entre 0.8 et 1.2.

4.5.2 CLASSIFICATION SELON LES RATIOS DE VITESSES

Après le calcul de ratios de vitesses de joueur humain et des bots, nous avons joint les résultats dans un fichier CSV qu'on a convertit en fichier ARFF pour classifier les données et déterminer s'il s'agit d'un joueur humain ou un bot.

L'importation de notre base de données sous *weka* donne :

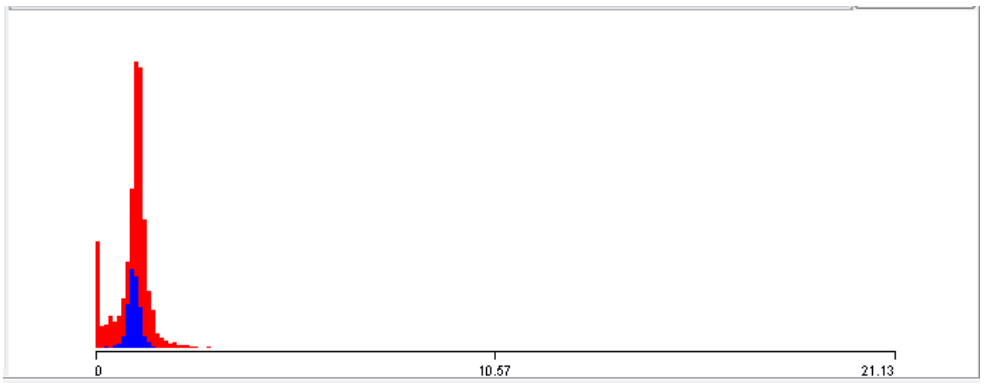

Figure 50 : Représentation de base de données de ratios sous weka

Nous avons 5395 instances où chaque instance et le ratio de vitesse sur un intervalle de 200 s. On voit sur la figure 50 deux couleurs ça veut dire qu'on a deux classes (Humain et bot). Pour mieux classifier ces données, nous avons appliqué l'algorithme J48 en utilisant cross validation =10 % (la définition de cross validation est en haut):

```
=== Classifier model (full training set) ===
J48 pruned tree
------------------
 ratio   <= 1.08
|    ratio   <= 0.82:   Humain (1205.0/105.0)
|    ratio    > 0.82:   Bot (1280.0/626.0)
 ratio    > 1.08:   Humain (2911.0/313.0)
Number of Leaves  : 3
Size of the tree :        5
Time taken to build model: 0.09 seconds
=== Stratified cross-validation ===
=== Summary ===
Correctly Classified Instances        4330               80.2446 %
Incorrectly Classified Instances      1066               19.7554 %
Kappa statistic                          0.3985
Total Number of Instances             5396
```

Figure 51: Classification des ratios avec J48

Les données ont été classifiées correctement avec un pourcentage de 80.24 % et avec un taux d'erreur de 19.76 %. Comme résultat de classification avec l'algorithme J48 on obtient l'arbre de décision suivant :

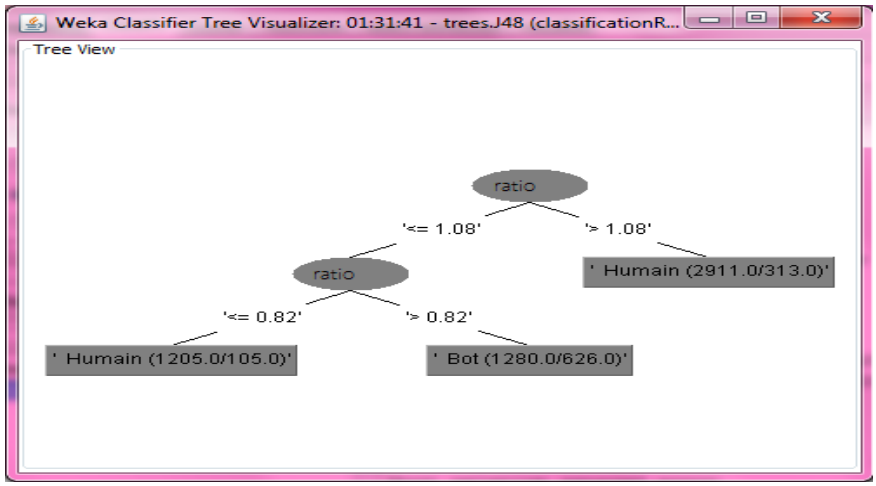

Figure 52: Arbre de décision de ratios

Cet arbre nous montre deux classes : Humain et bot.

D'après les valeurs qu'on a, l'algorithme J48 a fixé un premier seuil (la valeur selon lequel il va classifier). Notre seuil = 1.08 : Si ratio > 1.08 ➔ joueur humain et l'ajoute à la classe de l'humain sinon si ratio <=1.08 ➔ bot

Puis il a fixé un deuxième seuil = 0.82 : Si ratio <=0.82 ➔ joueur humain Sinon si ratio> 0.82 ➔ bot.

Donc, pour résumer, le ratio d'un joueur humain varie beaucoup et toujours inférieur à 0.82 ou supérieur à 1.08 (allant jusqu'à 3.051). Par contre, le ratio d'un bot ne varie pas beaucoup (toujours entre [0.82 et 1.08]).

Le ratio de vitesse est un autre critère de distinction entre les joueurs humains et les bots. En effet, en utilisant les résultats précédents, nous pourrons déterminer s'il s'agit d'un joueur humain ou d'un bot qui joue. Aussi nous pourrons savoir quand est ce que le joueur humain a joué lui-même et quand est ce qu'il a utilisé le bot. Cette méthode nous a permis de détecter les bots dans les jeux vidéo multi-joueurs en ligne en utilisant les techniques de forage de données.

4.6 VARIATION DE VITESSES

La variation de vitesse entre deux points a et b est la différence de la vitesse entre un point de départ et un point d'arrivée, divisée par la durée entre ces deux points c'est-à-dire :

Variation de vitesse = Vb-Va/Tb- Ta

Le calcul de variation de vitesses nous permet de déterminer le sens où la vitesse se fait: une accélération ou une décélération. En effet, l'agent se déplace sur la carte de jeu dans plusieurs directions et effectue plusieurs tâches : il tire ses ennemies, se cache, recule…

Dans cette expérience, nous avons utilisé une trace humaine, une trace de *Crbot*, une trace d'*Eraser* et une trace d'*Ice*. Nous avons utilisé les résultats de calcul des vitesses moyennes sur des intervalles de 200s, nous avons calculé la variation de vitesse entre ces intervalles. La durée Tb-Ta est toujours la même et est égale à 200 s puisque la trace est divisée en des intervalles de 200s.

Donc on calcule la différence de vitesse entre chaque deux intervalles successifs et on divise sur 200. Les figures suivantes montrent les variations de vitesse sur des intervalles de 200 s pour les agents :

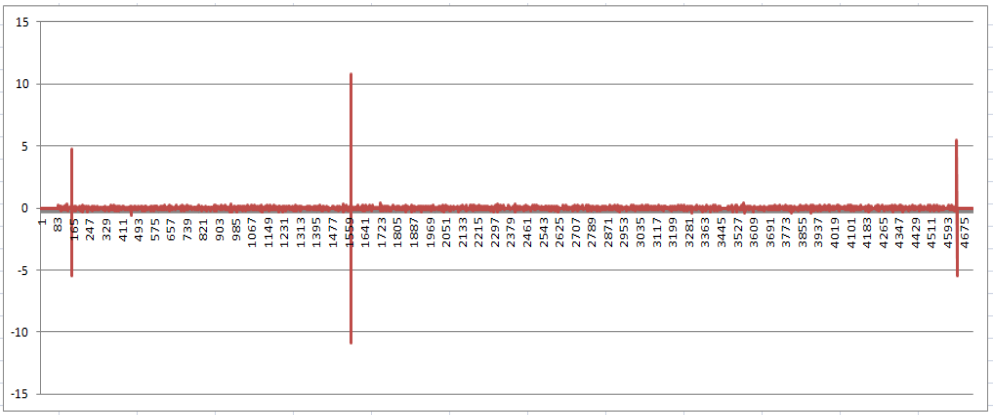

Figure 53: Variation de vitesse sur des intervalles de 200 s pour un joueur humain

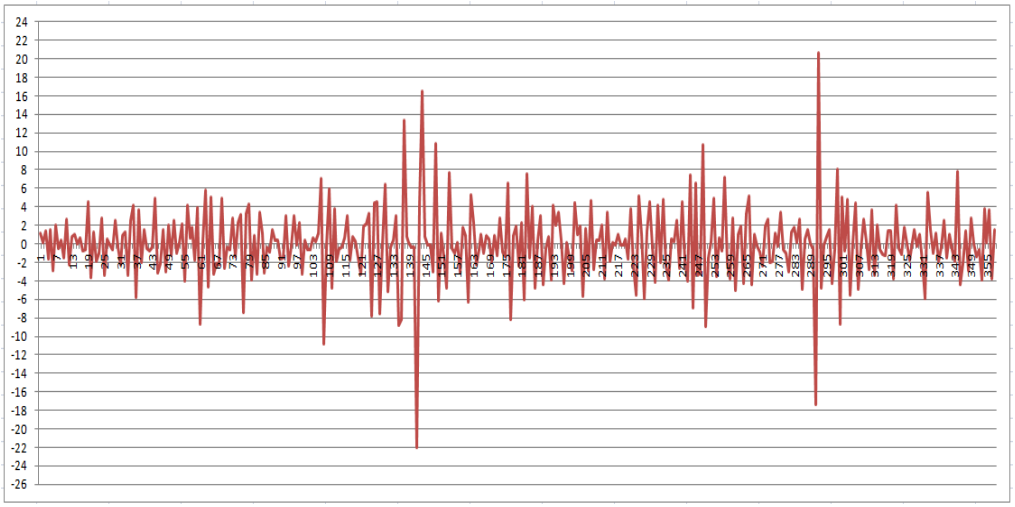

Figure 54 : Variation de vitesse sur des intervalles de 200 s pour Crbot

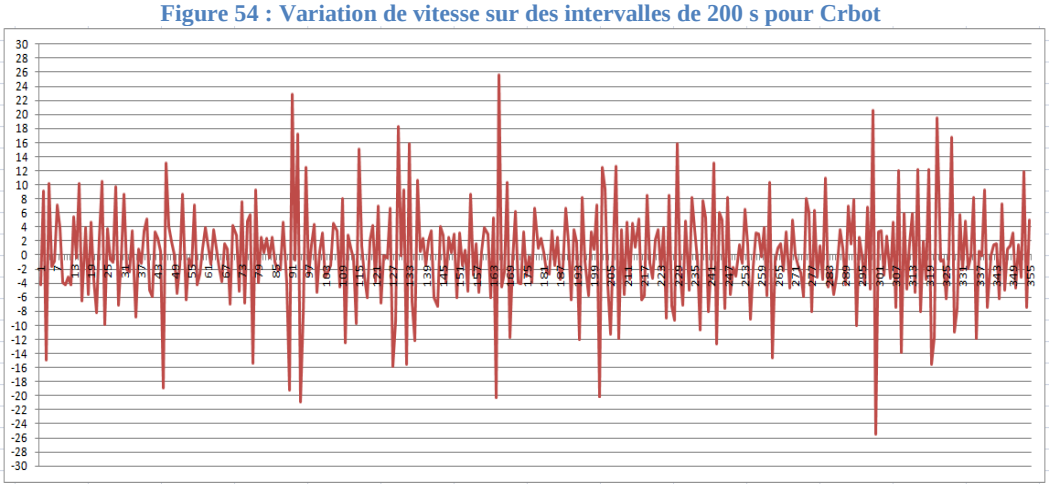

Figure 55: Variation de vitesse sur des intervalles de 200 s pour Eraser

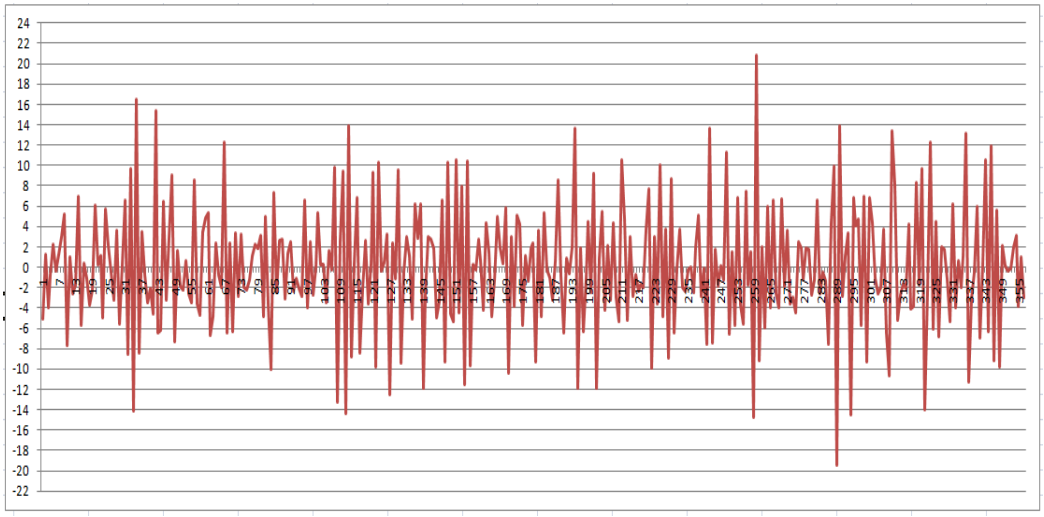

Figure 56: Variation de vitesse sur des intervalles de 200 s pour Ice

D'après les figures ci-dessus, nous remarquons que la vitesse varie beaucoup pour les bots alors que le joueur humain ne bouge pas beaucoup. Ceci est normal car les bots sont des programmes informatiques très rapides. Mais aussi, le joueur humain a fait quelques pics. En effet,

*entre l'intervalle 157 et 158, la vitesse a augmenté de 4.775 (passant de 0,735 à 5,508) puis elle a devenue presque nulle (0,009) ce qui entraine une diminution de -5,499.

*entre l'intervalle 1573 et 1574, la vitesse a augmenté de 10,752 (passant de 0,189 à 10,941) puis elle est devenue presque nulle (0,085) entrainant une diminution de -10,855.

*entre l'intervalle 4641 et 4642, la vitesse a augmenté de 5,475 (passant de 0 à 5,475) puis elle est revenue nulle entrainant une diminution de -5,475.

4.6.1 TRAVAIL PLUS DETAILLÉ SUR LAVARIATION DE VITESSES

Afin de mieux comprendre et analyser les variations de vitesses, nous avons supprimé les pics les plus remarquables et nous avons pris une partie de chaque courbe de variation de vitesse d'un agent :

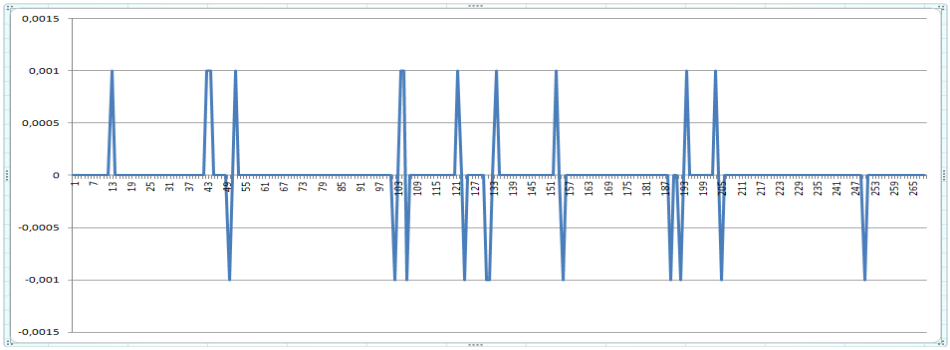

Figure 57: Partie de courbe de variations de Vitesse pour un joueur humain

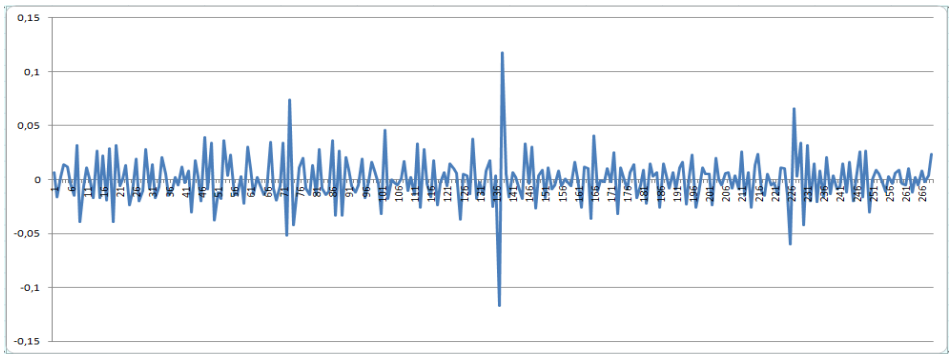

Figure 58: Partie de courbe de variations de vitesses pour Crbot

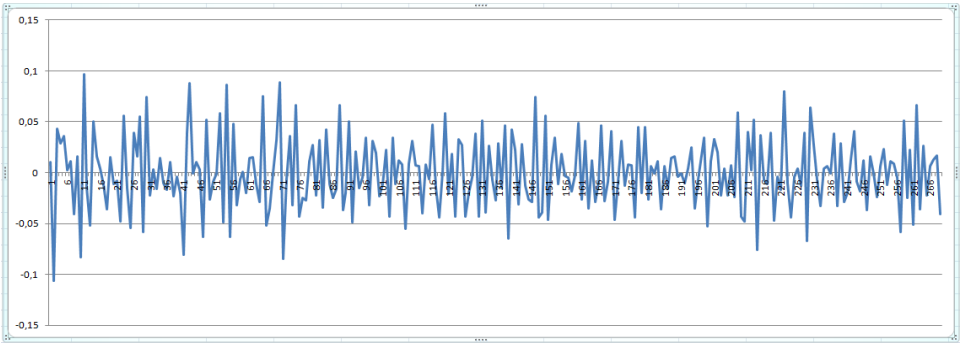

Figure 59: Partie de courbe de variations de vitesses pour Eraser

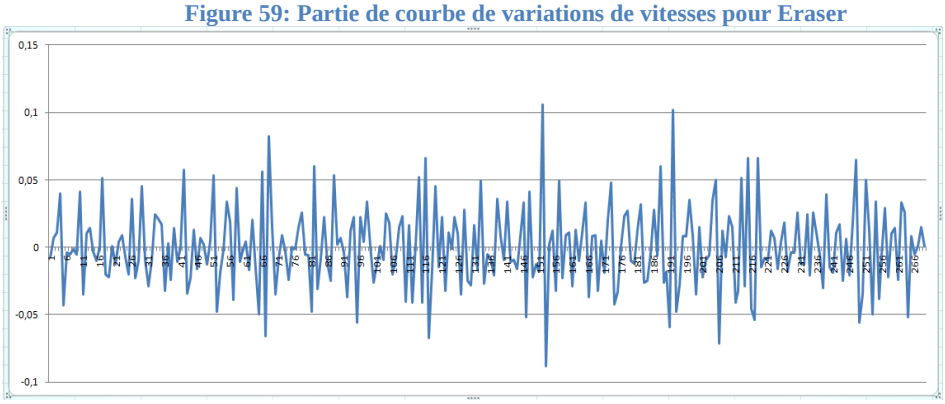

Figure 60: Partie de courbe de variations de vitesses pour Ice

D'après les courbes ci-dessus, nous remarquons que le joueur humain bouge beaucoup moins que les bots. En effet, il varie ou bien de 0.001 ou -0.001 ou il reste à 0. C'est-à-dire il fait une légère variation ou il reste stable. Mais sur la courbe entière de variation de vitesses de joueur humain, on voit des pics c'est-à-dire il reste presque stable pendant un bout de temps puis il accélère soudainement ensuite il décélère et retourne à son premier état. Ceci est dû à l'hésitation de joueur humain déjà un être humain est hésitant de nature et ça se voit même dans sa façon de jouer. En effet, le temps où il reste stable ou il fait une légère variation de sa vitesse c'est le temps où il est en train de regarder et de réfléchir avant la prise de sa décision. Puis après son accélération brusque, il revient stable puisque il est en train de réagir par rapport à sa nouvelle position et de préparer sa nouvelle action.

Par contre, pour les bots, puisque ils ne réfléchissent pas, on ne voit pas sur leurs courbes de vitesses des périodes de stabilité. Ils sont toujours actifs. Ils savent bien réagir, en toute position, où ils

sont. Donc ils n'ont pas besoin de prendre un temps de break pour décider quoi faire prochainement. De plus, leurs vitesses sont toujours élevés, c'est pour cela leurs pics ne sont pas remarquables autant qu'un joueur humain.

4.6.2 CLASSIFICATION SELON LES VARIATIONS DE VITESSES

Les résultats obtenus suite au calcul de variations de vitesses nous montrent des différences entre les comportements de joueur humain et des bots et nous permettent de distinguer entre le joueur humain et le bot en analysant les courbes. C'est pour cela nous passons à la classification de données afin de trouver un seuil avec lequel nous pourrons détecter automatiquement s'il s'agit d'un joueur humain ou bot.

Nous avons 5394 instances. Nous avons importé notre base de données sous *weka* pour la classifier :

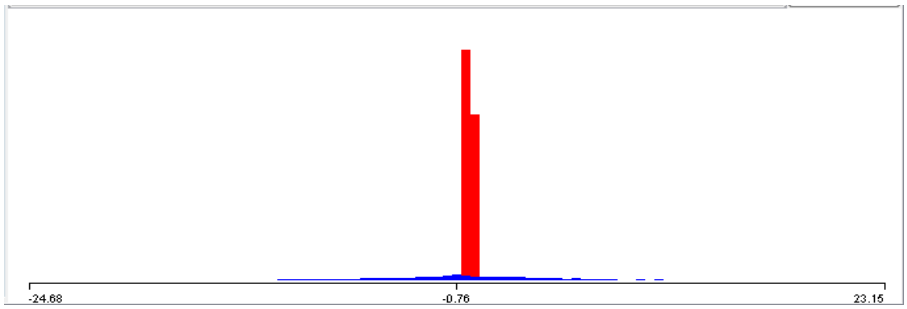

Figure 61: Représentation de données de variations de vitesses sous weka

La représentation de données sous *weka* nous montre 2 classes (2 couleurs). Les instances ont des valeurs allant de -24.68 jusqu'à 23.15.

REPTree est un algorithme de classification supervisée de données. Cet algorithme utilise la logique des arbres de régression (arbres de décision) et crée plusieurs arbres dans plusieurs itérations. Après ça, il sélectionne le meilleur des arbres générés. Ce dernier représente le résultat de classification de nos données. *REPTree* construit l'arbre de décision en utilisant le gain d'information comme critère de fonctionnement.

```
REPTree
============
 variation   < -0.49 :   Bot (347/3) [177/2]
 variation  >= -0.49
|    variation   < 0.57 :   Humain (2929/51) [1475/38]
|    variation   >= 0.57 :   Bot (320/1) [146/3]
 Size of the tree : 5
Time taken to build model: 0.14 seconds
=== Stratified cross-validation ===
=== Summary ===
Correctly Classified Instances        5289               98.0534 %
Incorrectly Classified Instances      105                1.9466 %
Kappa statistic                          0.9375
```

Figure 62 : Classification supervisée des variations de vitesse avec *REPTree*

Les données ont été classées correctement à 98.18% avec un taux d'erreur de 1.81 %.

Comme résultat de l'application de l'algorithme *REPTree*, nous avons obtenu l'arbre de décision ci-dessous :

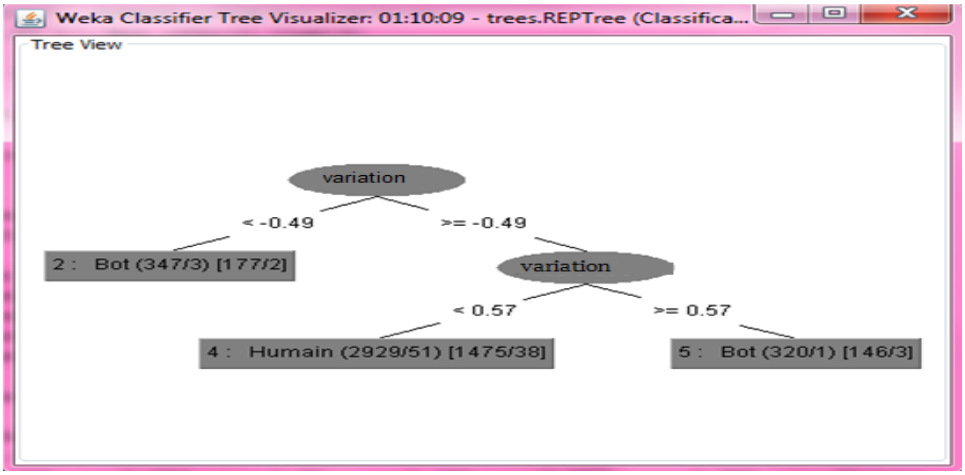

Figure 63: Arbre de décision de variations de vitesses

En effet, si la variation de vitesse est inférieure à -0.49 ou supérieur ou égale à 0.57 alors il s'agit d'un bot. Sinon, si variation >= -0.49 et inférieure à 0.57, alors il s'agit d'un joueur humain.

Le taux d'erreurs est expliqué par les pics que le joueur humain a faits. Par exemple une instance humaine de valeur =5 ne peut pas faire partie de la classe Humain, parce que la classe

humaine a des valeurs qui sont inclus entre -0.49 et 0.57, donc cette instance a été classée dans la classe Bot ce qui entraine le taux d'erreur.

Suite à cette expérimentation, nous pouvons conclure que la variation de vitesse est un critère qui nous permet de détecter les bots dans les jeux vidéo en ligne. En effet, en regardant la courbe de variation de vitesses d'un agent nous pouvons conclure s'il s'agit d'un bot ou d'un joueur humain. Cette méthode est efficace mais l'utilisation de forage de données nous facilite plus la tâche et nous permet de détecter les bots dans les jeux vidéo multi-joueurs en ligne et nous permet, en analysant les traces de jeux, de savoir quand est ce que le joueur humain a joué et quand est ce que il a utilisé un bot.

4.7 ACCELERATION MOYENNE DE LA TRACE

Après avoir calculé les variations de vitesses, pour chaque trace, sur des intervalles de 200s, nous avons passé au calcul de l'accélération moyenne.

L'accélération moyenne permet de connaitre la variation de vitesse en valeur absolue et correspond à un taux moyen de variation de la vitesse sur un intervalle du temps. La formule de l'accélération moyenne est :

Accélération moyenne = (1/ nombre de variations) * ∑ |variation de vitesses|

Dans ce contexte, nous avons calculé la somme de valeurs absolues de variations de vitesses puis nous avons divisé cette somme sur le nombre de variations pour avoir à la fin la valeur de l'accélération moyenne. Nous avons joint tous les résultats dans un fichier CSV qu'on a convertit à un fichier ARFF pour l'utiliser sous *weka*.

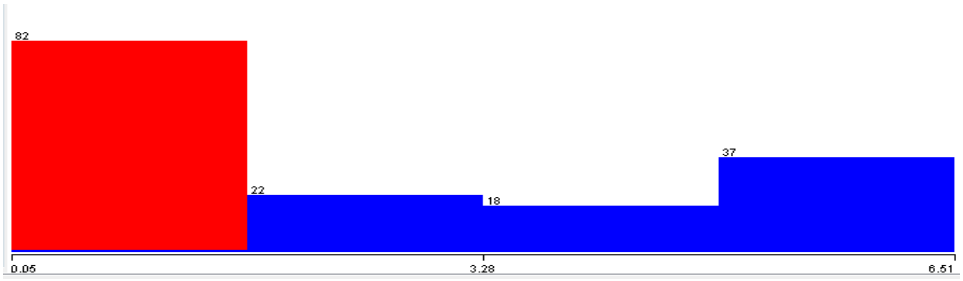

Figure 64: Représentation de données de l'accélération moyenne sous *weka*

le montre la figure 64, les valeurs de l'accélération moyenne de traces de bots et des joueurs humains varient entre 0.05 et 6.51 . Aussi, ça se voit qu'on a 2 classes.

En total, nous avons 159 instances. Suite à une classification supervisée avec l'algorithme *J48* et en utilisant un cross-validation de 10 %, nous avons obtenu :

```
J48 pruned tree
------------------
 acceleration   <= 0.62:   Humain (81.0)
 acceleration   > 0.62:   Bot (78.0)
Number of Leaves  :     2
Size of the tree :          3
Time taken to build model: 0.02 seconds
=== Stratified cross-validation ===
=== Summary ===
Correctly Classified Instances         158               99.3711 %
Incorrectly Classified Instances        1                0.6289 %
Kappa statistic                          0.9874
```

Figure 65 : Classification supervisée des accélérations moyennes avec J48

Les données ont été à 99.37 % classées correctement avec un taux d'erreur presque nul. Comme résultat de classification supervisée de nos données en utilisant l'algorithme *J48*, nous avons obtenu l'arbre de décision ci-dessous. Les données ont été classée sur deux classes : Humain et bot. Le seuil de classification est 0.62. En effet, si l'accélération moyenne est inférieure ou égale à 0.62, alors il s'agit d'un joueur humain. Sinon, si l'accélération est supérieure à 0.62 alors il s'agit d'un bot.

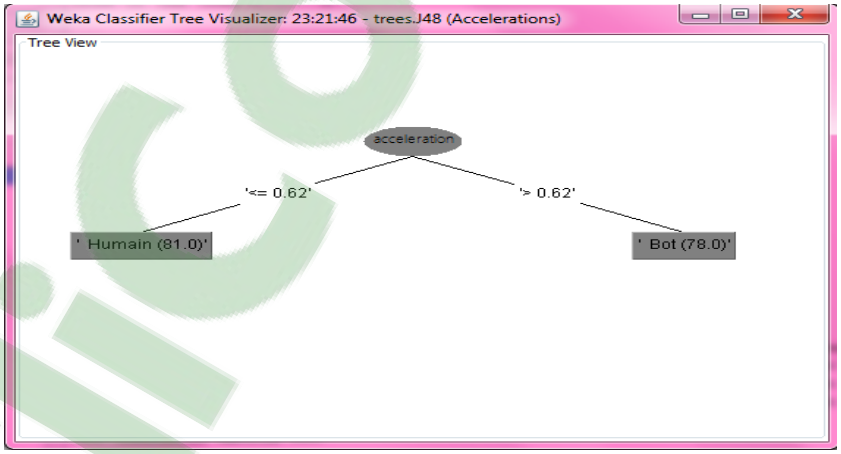

Figure 66: Arbre de décision des accélérations moyennes

Suite à cette expérience, nous pouvons conclure que l'accélération moyenne est un autre critère pour faire la différence entre les bots et les joueurs humains ce qui nous aide à détecter automatiquement les bots dans les jeux vidéo multi-joueurs en ligne.

4.8 ECART TYPE ABSOLU DE LA TRACE

Après avoir calculé les variations de vitesses sur des intervalles de 200s ainsi que l'accélération moyenne de la trace, nous avons passé au calcul de l'écart type de la trace.

L'écart type absolu sert à mesurer la dispersion d'un ensemble de données. Il se définit comme égale à la moyenne des valeurs absolues des différences entre les données et leur moyenne :

$$\mathbf{Em = (1/N) * \sum |Xi- Xmoyen|}$$

Tels que : em est la valeur de l'écart type absolu, N est l'ensemble de données, Xi est un nombre réel qui appartient à N, Xmoyen est la déviation moyenne.

Si on applique cette formule sur notre ensemble de données, elle deviendra :

$$\mathbf{Ecart\ Type\ Absolu = (1/nombre\ de\ variations) * \sum |variation\ de\ vitesses- accélération\ moyenne|}$$

Donc, pour chaque trace nous avons calculé son écart type. Puis, nous avons stocké les résultats sous un fichier CSV qu'on a été convertit en fichier ARFF pour l'utiliser sous weka.

Suite à l'importation de notre base de données sous weka, on a obtenu :

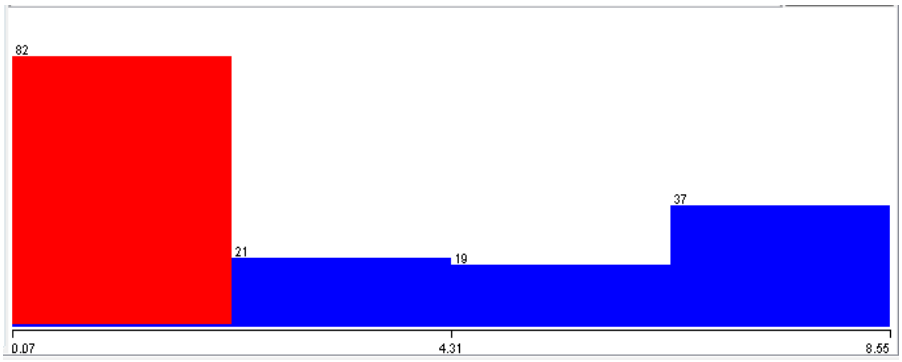

Figure 67 : Représentation de données de l'écart Type sous weka

Au total nous avons 159 instances. On applique l'algorithme *J48* sur nos données afin d'établir une classification supervisée :

```
J48 pruned tree
------------------
 ecartType   <= 1.11:   Humain (81.0)
 ecartType   > 1.11:   Bot (78.0)
Number of Leaves  :     2
Size of the tree :          3
Time taken to build model: 0 seconds
=== Stratified cross-validation ===
=== Summary ===
Correctly Classified Instances         158               99.3711 %
Incorrectly Classified Instances        1                0.6289 %
Kappa statistic                          0.9874
```

Figure 68: Classification supervisée des écarts types avec J48

Ces instances en été classées sous 2 classes : Humain et Bot. Les données ont été à 99.37 % classées correctement. Ça donne l'arbre de décision ci-dessous :

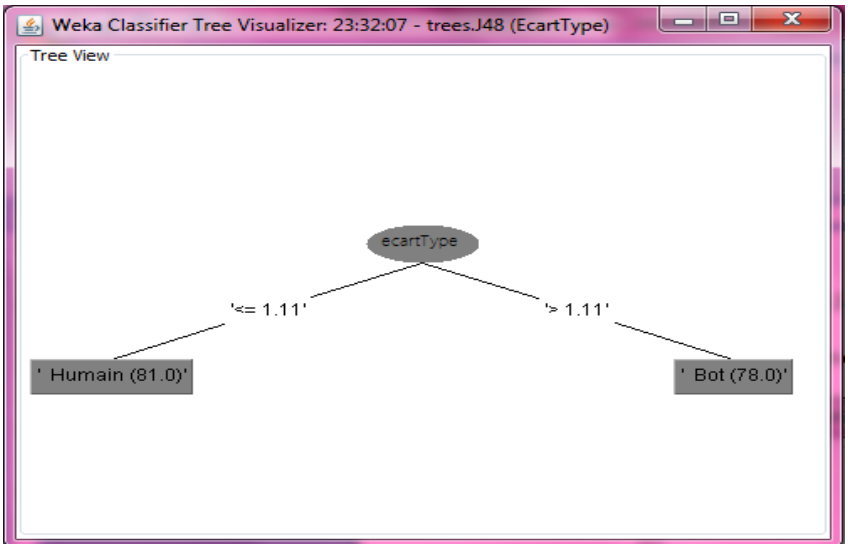

Figure 69: Arbre d décision des écarts types

En effet, si l'écart Type d'une trace est inférieur ou égal à 1.11 alors on conclut qu'un s'agit d'un joueur humain sinon si l'écart type est supérieur à 1.11, alors il s'agit d'un bot.

Comme conclusion pour cette expérience, nous avons constaté que l'écart type est un autre critère pour faire la différence entre les bots et les joueurs humains. Donc en évaluant l'écart type d'une trace, nous pourrons savoir s'il s'agit d'une trace humaine ou une trace de

bot. En effet, ceci permet une détection automatique des bots dans les jeux vidéo multi-joueurs en ligne.

4.9 LONGUEURS D'ONDES POUR LES TRACES

Au cours des expériences précédentes nous avons travaillé sur la vitesse moyenne de la trace, la vitesse sur des intervalles de 200s ainsi que les variations de vitesses sur des intervalles de 200s et nous avons classés nos données en se basant sur les résultats de celles-ci. Mais la vitesse peut être facilement modifiée dans le code, par les programmeurs des bots. C'est pour cela nous avons pensé à travailler sur d'autres métriques. Parmi ces métriques on distingue la longueur d'onde. En effet, la longueur d'onde est la distance séparant deux sommets dans la courbe de variations de vitesses. L'objectif de calculer la longueur d'onde dans notre projet est de déterminer la période moyenne qui sépare deux augmentations de vitesses c'est-à-dire déterminer, en moyenne, la période après laquelle un agent accélère.

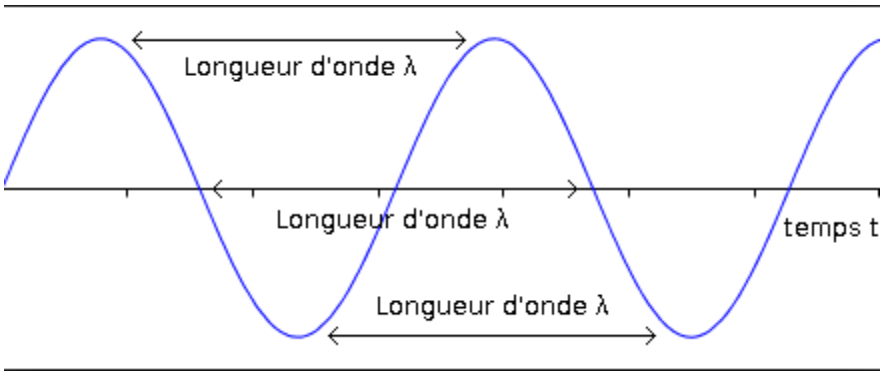

Figure 70: Longueur d'onde [montpe.weebly.com]

La formule avec laquelle nous avons calculé les longueurs d'ondes est :

Longueur d'onde = position de pic (i+1) – position de pic (i)

Nous avons stocké toutes les positions de sommets de la courbe de variations de vitesses sur des intervalles de 200s, puis nous avons calculé leurs longueurs d'ondes. Nous avons utilisé quatre traces : une trace humaine, une trace de CRbot, une trace d'Eraser et une trace d'Ice.

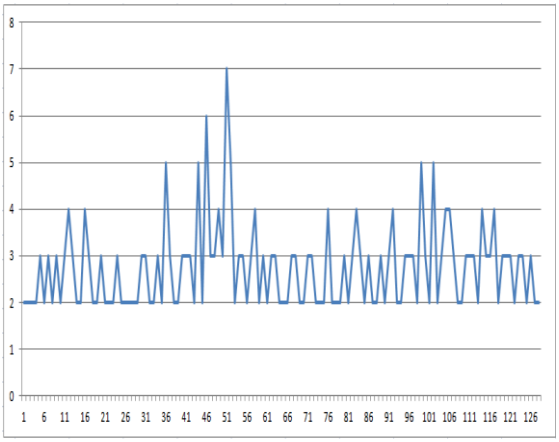

Figure 71 : Longueurs d'ondes pour CRbot

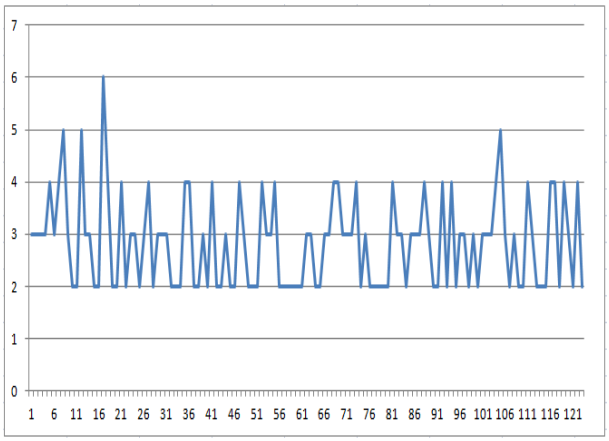

Figure 72 : Longueurs d'ondes pour Eraser

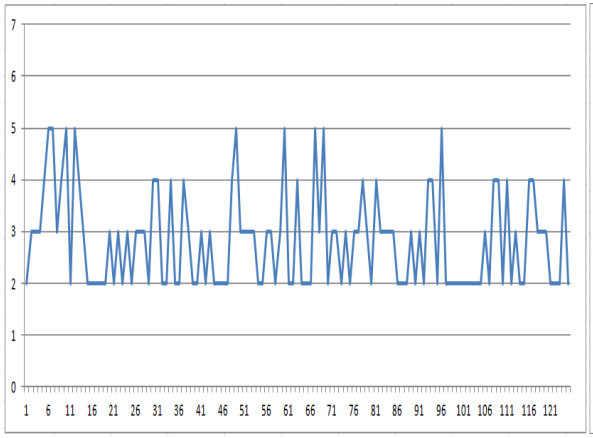

Figure 73: Longueurs d'ondes pour ICE

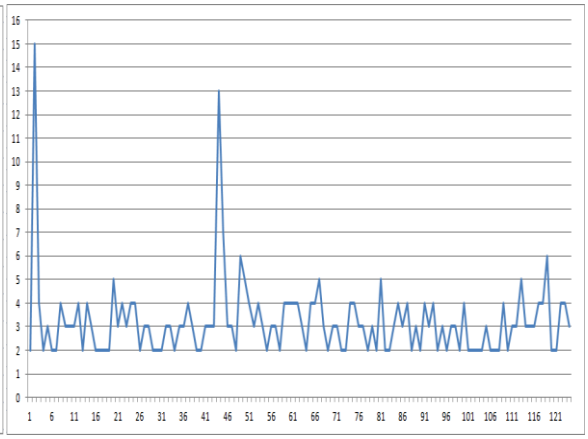

Figure 74: Longueurs d'ondes pour joueur humain

Si longueur d'onde =1 c'est-à-dire que position de pic (i+1) – position de pic (i) =1 = 200 s

Si longueur d'onde =2 c'est-à-dire que position de pic (i+1) – position de pic (i) =2 = 2*200 s = 400 s

Et ainsi de suite.

La figure 74 montre la variation de longueurs d'ondes pour le joueur humain. Il a commencé par une longueur d'onde égale à 15, ça veut dire que le joueur a resté stable sur une longue période (15*200= 3000 s = 50 min). Dans cette courbe, il existe déjà 2 grands pics (15 et 13). La majorité de longueurs d'onde varient entre 2 et 4. Ça change beaucoup. Ce n'est pas du tout périodique.

Par contre dans les courbes 71,72 et 73, nous remarquons qu'il existe de la périodicité c'est-à-dire il y'a de l'alternance qui se voit dans les courbes de variation de longueurs d'ondes pour les bots. Par exemple, dans la figure 73, dans la courbe de variation de longueurs d'ondes d'*Ice*, nous voyons que le bot a une longueur d'onde égale à 3 ; trois fois successivement, juste après, il a fait 3 fois une longueur d'onde égale à 4, trois fois successivement, c'est-à-dire qu'il fait une accélération 3 fois

successivement chaque 600 s puis il a fait une accélération chaque 800 s 3 fois successivement puis il a fait une longueur d'onde égale à 3, deux fois successifs encore et ainsi de suite.

Aussi, dans la courbe d'*Eraser*, nous avons remarqué qu'il fait successivement, en alternance une langueur d'onde égale à 4 puis une longueur égale à 3 puis une autre longueur d'onde égale à 3 puis 4 et ainsi de suite.

Pour les quatre agents, les valeurs de longueurs d'ondes varient entre 3 et 4 ou 5. Il existe quelques valeurs différentes entre les deux types d'agents mais la majorité de valeurs sont les mêmes

Nous avons importé notre base de données sous weka :

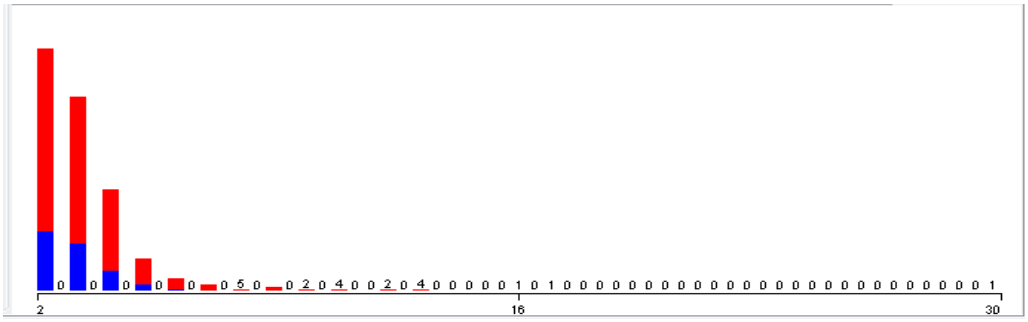

Figure 75: Représentation de données de longueur d'onde sous weka

Comme le montre la figure 75 les valeurs de écarts types pour les deux types d'agents sont les mêmes, donc une classification supervisée ou non supervisée des données n'est pas possible. En effet, les instances ont les mêmes valeurs pour les deux classes : 2, 3, 4,5. C'est pour cela dans cette expérience, *weka* n'est pas capable de classifier les données automatiquement.

Ce qu'on peut conclure suite à cette expérience, est que les valeurs de longueurs d'ondes ne font pas la différence entre un joueur humain et un bot. Ici ce qui fait la différence entre les deux types d'agents c'est la périodicité c'est-à-dire l'organisation des bots en faisant leurs accélérations vu qu'ils sont des programmes et ce qui n'existe pas chez un joueur humain. Donc ici ce ne sont pas les valeurs de l'écart type qui permettent de détecter les bots dans les jeux vidéo multi-joueurs en ligne mais c'est la périodicité qui existe dans la courbe des écarts types de bots et pas chez les joueurs humains c'est-à-dire ce sont les accélérations qui sont organisées dans le parcours du bot et qui n'est pas organisé dans le parcours de joueur humain.

4.10 ONDES MOYENNES DES TRACES

L'onde moyenne de la trace d'un agent est la valeur moyenne d'une onde qu'un agent peut faire. Autrement dit, c'est la période moyenne après laquelle un agent fait une accélération. Pour calculer l'onde moyenne d'une trace nous avons utilisé la formule suivante :

Onde moyenne = (1/nombre des ondes) * ∑ (valeurs d'ondes sur des intervalles de 200s)

Après avoir calculé les longueurs d'ondes moyennes pour les traces humaines et les traces des bots, nous avons joint nos données, puis, nous avons importé nos donnés sous weka :

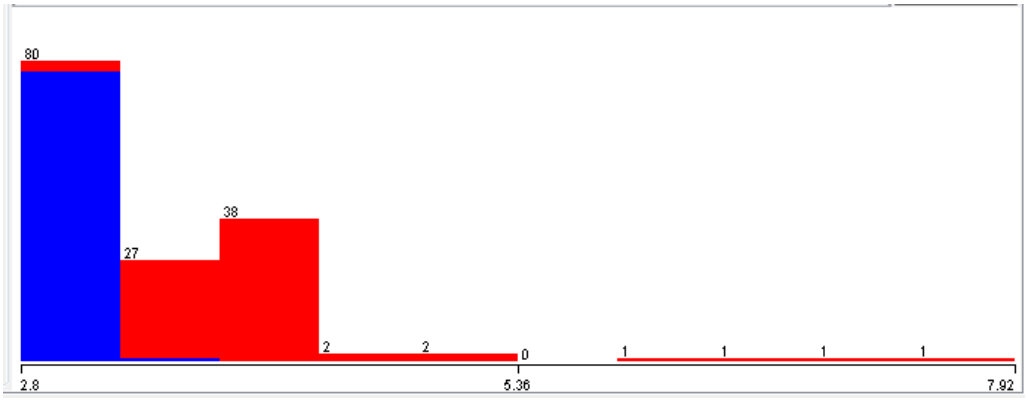

Figure 76: Représentation de données des ondes moyennes sous weka

Comme le montre la figure 76 ci-dessus, les valeurs des ondes moyennes varient entre 2.8 et 7.92, nous voyons aussi deux classes. Une classification supervisée avec *J48* (cross validation = 10%) donne :

```
J48 pruned tree
------------------
 OndeMoy   <= 3.34:   Bot (82.0/4.0)
 OndeMoy   > 3.34:   Humain (71.0)
Number of Leaves  :          2
Size of the tree :       3
Time taken to build model: 0.04 seconds
=== Stratified cross-validation ===
=== Summary ===
Correctly Classified Instances         147               96.0784 %
Incorrectly Classified Instances         6               3.9216 %
Kappa statistic                          0.9215
```

Figure 77 : Classification supervisée des ondes moyennes avec J48

Le taux des instances correctement classées est 96.08 % avec un taux d'erreur de 3.92 %.

Suite à la classification supervisée avec l'algorithme *J48* de nos données, nous avons obtenu l'arbre de décision suivant comme résultat de classification :

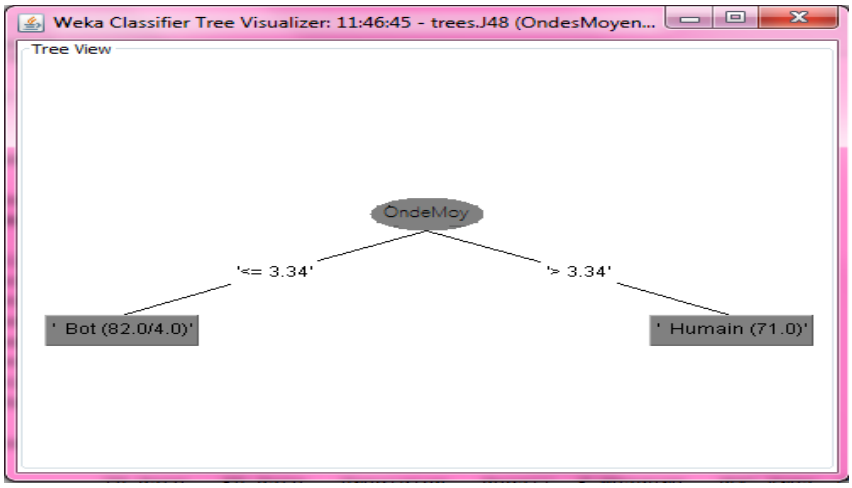

Figure 78: Arbre de décision des ondes moyennes

D'après notre arbre, si la longueur moyenne d'une onde est inférieure ou égale à 3.34 alors on déduit qu'il s'agit d'un bot sinon si la longueur moyenne d'une onde est supérieure à 3.34, alors on déduit qu'il s'agit d'un joueur humain.

A partir de la longueur d'onde moyenne d'une trace, nous pourrons déduire s'il s'agit d'une trace humaine ou d'une trace d'un bot. Et par la suite, ça nous aidera à détecter les bots dans les jeux vidéo multi-joueurs en ligne en utilisant les techniques de forage de données.

Suite à ça, nous avons calculé l'écart types des longueurs d'ondes. Nous avons stocké nos résultats dans un fichier ARFF et nous avons importé notre base de données sous *weka* :

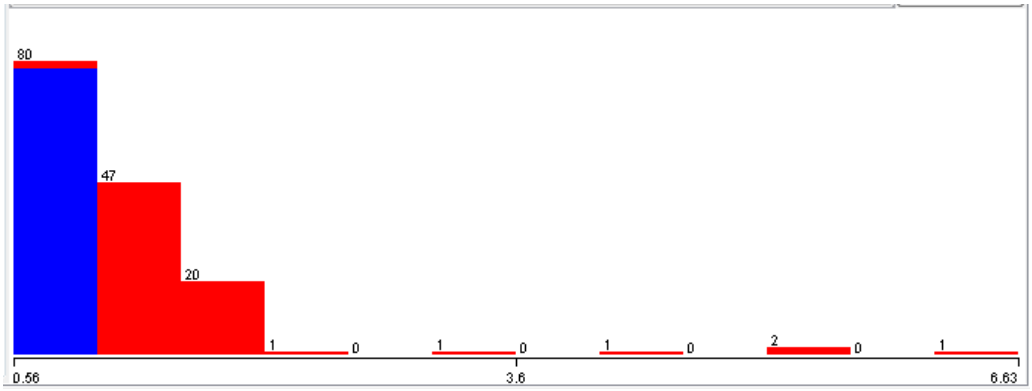

Figure 79: Écart Type des longueurs d'ondes moyennes

Les données varient entre 0.56 et 6.63. Suite à une classification supervisée de longueurs d'ondes moyennes, en utilisant l'algorithme *J48* avec cross validation = 10 %, nous avons obtenu :

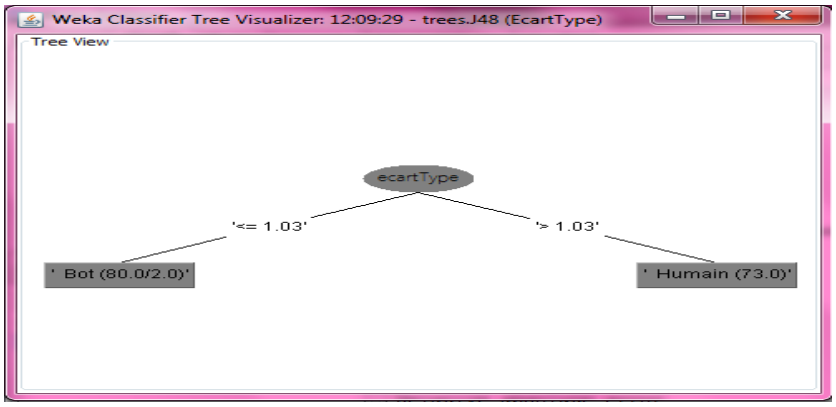

Figure 80: Écarts types de longueurs d'ondes moyennes de traces

Suite à une classification supervisée avec l'algorithme *J48*, les données ont été classées correctement à 97 % avec un taux d'erreur de 3 %. Comme résultat de classification nous avons eu un arbre de décision avec 2 classes : Bot et Humain.

En, effet, si l'écart type est inférieur ou égale à 1.03, alors il s'agit d'un bot, sinon si l'écart type est supérieur à 1.03 alors il s'agit d'un joueur humain.

L'écart type des ondes moyennes peut être un autre critère avec le quel on fait la différence entre un joueur humain et un bot et ceci nous permet de détecter les bots automatiquement dans les jeux vidéo multi-joueurs en ligne, en se basant sur les techniques de forage de données.

4.11 POURCENTAGES DE VITESSES PAR RAPPORT A LA VITESSE MOYENNE

Le pourcentage de vitesses est le ratio de vitesse. Donc après avoir calculé le pourcentage de vitesses pour chaque trace ainsi que le Ratio moyen (**ratio moyen = (1/nombre de ratios) *∑ (ratio de vitesses)**), nous avons calculé l'écart type de ratios de vitesses en appliquant la formule suivante :

Ecart Type Absolu = (1/nombre de ratios) *∑ |ratio de vitesses- ratio moyen|

L'importation de notre base de données sous *weka* donne :

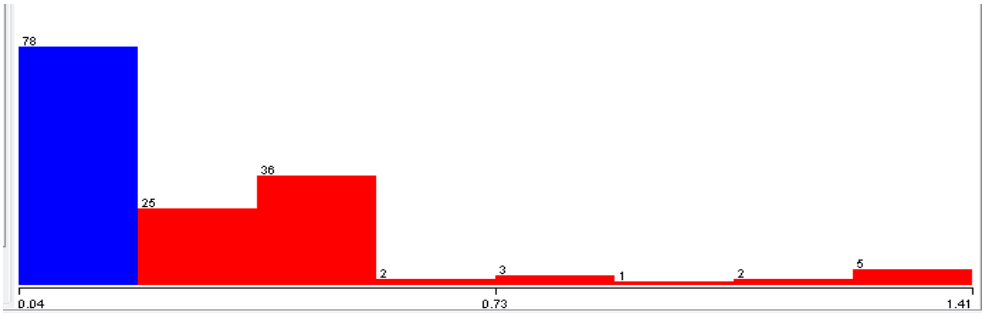

Figure 81: Représentation de base de données de données de l'écart type de ratios de vitesses sous weka

Les instances varient entre 0.04 et 1.4. En appliquant l'algorithme *J48*, nous avons eu comme résultat :

```
J48 pruned tree
------------------
 ecartType   <= 0.19:   Bot (78.0)
 ecartType   > 0.19:   Humain (74.0)
Number of Leaves  :     2
Size of the tree :          3
Time taken to build model: 0.01 second
=== Stratified cross-validation ===
=== Summary ===
Correctly Classified Instances         152               100      %
Incorrectly Classified Instances         0                0      %
```

Figure 82: Classification supervisée des écarts types de pourcentages de vitesses avec J48

Les instances ont été classées à 100 % correctement avec 0% comme taux d'erreur. Suite à une classification supervisée avec *J48*, nous avons obtenu un arbre de décision constitué de 2 classes : Humain et Bot.

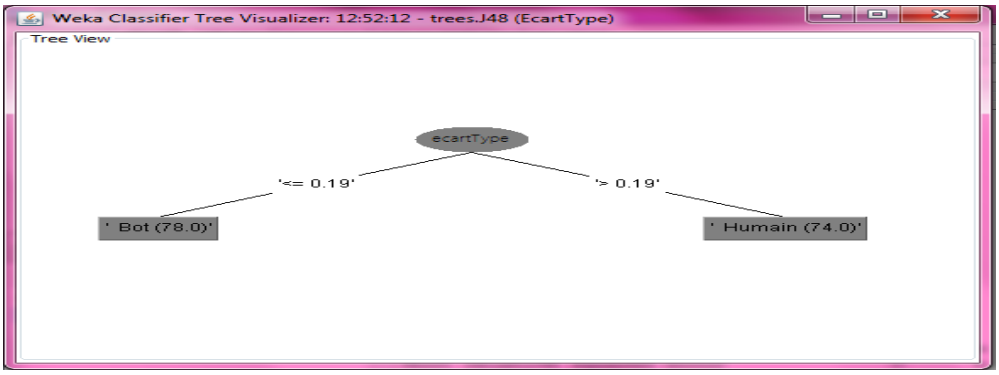

Figure 83: Arbre de décision des écarts types absolus de pourcentages de vitesses

D'après l'arbre de décision obtenu, notre seuil de classification est 0.19. Donc si écart type est inférieur ou égal à 0.19 alors il s'agit d'un bot sinon si l'écart type est supérieur à 0.19 alors il s'agit d'un joueur humain.

Suite à cette expérience, nous pouvons conclure qu'en se basant sur les valeurs des écarts types absolus de pourcentages de vitesses, nous pouvons déduire s'il s'agit d'un joueur humain ou d'un bot. En effet, ça nous aide à détecter les bots dans les jeux vidéo multi-joueurs en ligne.

Suite à ça, nous avons passé au calcul les longueurs d'ondes sur des intervalles de 200s pour les pourcentages de vitesses, en appliquant la formule utilisée précédemment. Pour ce fait, nous avons utilisé une trace humaine, une trace de *Crbot*, une trace d'*Eraser*, et une trace d'*Ice* :

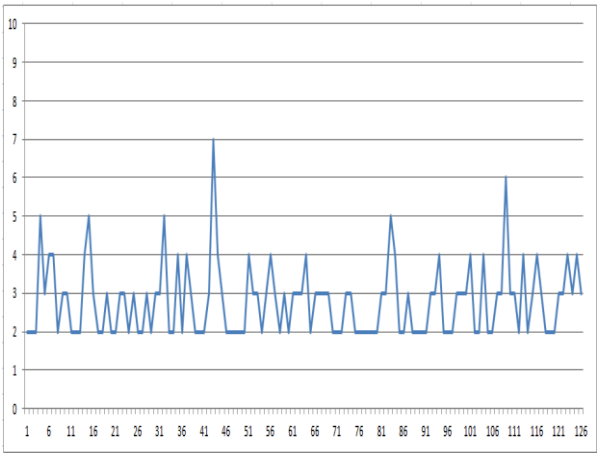

Figure 84 : Longueurs d'ondes pour CRbot

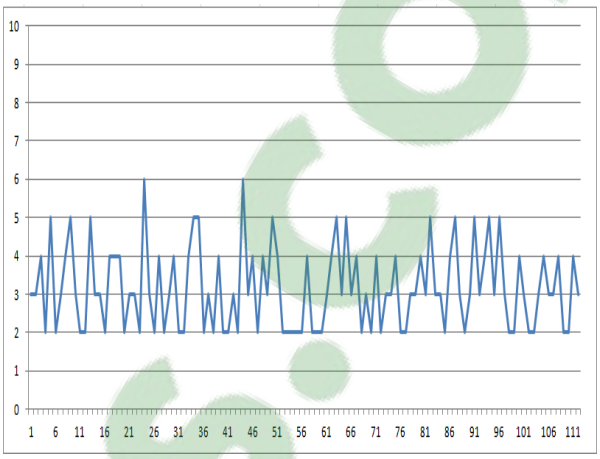

Figure 85: Longueurs d'ondes pour Eraser

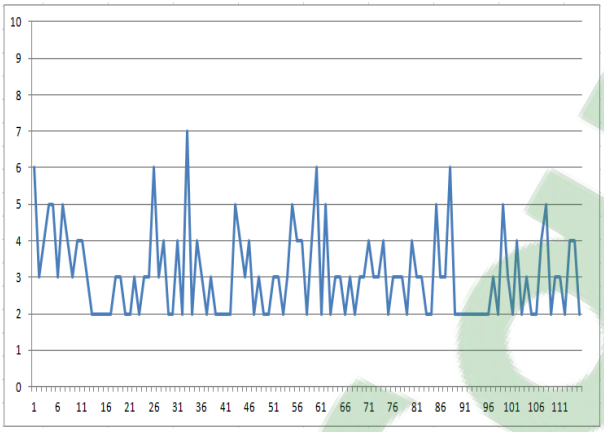

Figure 86 : Longueurs d'ondes pour Ice

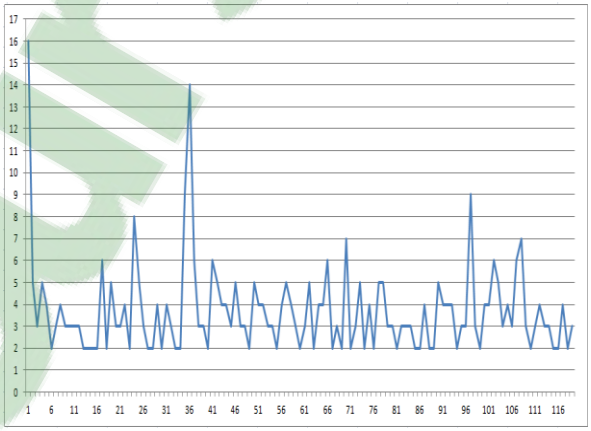

Figure 87: Longueurs d'ondes pour un joueur Humain

Pour commencer, nous remarquons, d'après les courbes de variations de longueurs d'ondes de pourcentages de vitesses, que le joueur humain reste plus stable c'est-à-dire il ne bouge pas pendant un certain temps, avant de faire une accélération. En effet, il a des valeurs élevés de longueurs d'ondes allant jusqu'à 16 alors que pour les bots les valeurs de longueurs d'ondes ne dépassent pas 7 et les valeurs varient généralement entre 2 et 5.

La mesure de longueurs d'ondes pour les pourcentages de vitesse peut nous permettre de faire la différence entre un joueur humain et un bot. Mais ça ne nous permette pas de détecter automatiquement les bots dans les jeux vidéo multi-joueurs en ligne en se basant sur les valeurs de longueurs d'ondes. En effet, une analyse des courbes est nécessaire pour déterminer s'il s'agit d'un joueur humain ou d'un bot.

4.12 LES MODÈLES DES AGENTS

Dans les sections précédentes, nous avons montré les résultats des expériences réalisées dans le but de détecter les différences entre un joueur humain et un bot. Nous avons trouvé finalement des différences majeurs qui peuvent nous aider à faire la différences entre un joueur humain et un bot, et par la suite détecter les bots dans les jeux vidéo multi-joueurs en lignes, tout en détectant les profils similaires à celui de bot.

Lors des expériences précédentes, les données ont été bien classées, mais quand même il y avait un petit taux d'erreur. Ces expériences nous ont donné des meilleurs résultats avec lesquels nous pourrons déterminer s'il s'agit d'une trace humaine ou celle d'un bot.

Les critères sur lesquels, la construction de nos profils a été basée, sont :

La vitesse moyenne de la trace = vitesseTot

L'accélération moyenne de la trace = Accelerations

L'écart type de la trace = Ecart Type

Les ratios de vitesse = Ratios

Les ondes moyennes = OndesMoyennes

4.12.1 LE RESEAU DE NEURONES

Le réseau de neurones est un algorithme d'apprentissage automatique. Il est consacré pour la recherche scientifique. En effet, ce dernier est basé sur la transmission nerveuse de cerveau de l'humain. Ceci a pour rôle d'élaborer l'information reçue et de transmettre les résultats à d'autres neurones.

Le cerveau humain développe mieux les solutions intelligentes qu'un bot, cependant ce dernier est beaucoup plus rapide. En vue de traitement de l'information, le bot applique les algorithmes et exécute les instructions qui font partie de sa programmation alors qu'un joueur humain réfléchit, décide puis il réagit. De plus les bots sont programmés par des êtres humains et même si les bots fonctionnent en utilisant les techniques de l'intelligence artificielle, les êtres humains restent les plus intelligents car tout est inventé par l'homme.

On utilise les réseaux de neurones afin de résoudre les problèmes de classification, d'extraction des informations, des données… c'est un ensemble des règles et de traitement de données permettant de construire un modèle de comportement à partir de données qui sont des exemples de ce comportement.

Dans ce cadre, nous avons appliqué l'algorithme *MultilayerPerceptron* implémenté sous weka, sur nos données. Il s'agit d'un réseau de neurones de type *Perceptron multicouche* qui est le plus utilisé dans le réseau de neurones artificiel.

Suite à l'application de l'algorithme *MultilayerPerceptron*, nous avons obtenu comme résultat :

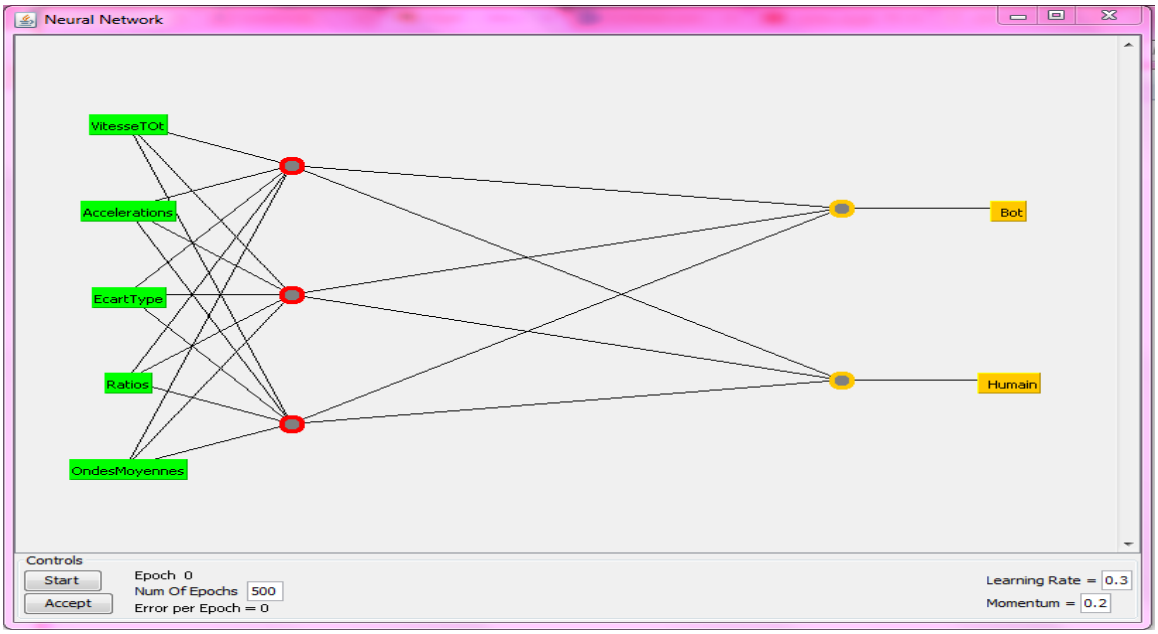

Figure 88 : Représentation de nos modèles avec l'algorithme *MultilayerPerceptron*

La figure 88 montre :

-Notre couche d'entrée constituée de : vitesseTot, Acceleretions, EcartType , Ratios, OndesMoyennes

-La première couche cachée (les points rouges)

-La deuxième couche cachée (les points jaunes)

-La couche de sortie : ce sont nos modèles (Bot et Humain)

4.12.2 L'ALGORITHME DBSCAN

L'utilisation de mesure de similarité dans les algorithmes *J48, REPTree* et *Simple K-Means*, est efficace, mais elle est moins efficace que la densité de voisinage [18]. La densité est le nombre de points à l'intérieur d'un rayon donné. La notion de *clustering* basée sur la densité consiste à trouver les points(les instances) ayant un voisinage de cardinalité élevée, trouver les voisins de ces points et les regrouper dans un cluster. L'algorithme *DBSCAN* (Density-Based Spatial Clusturing Applications with Noise) est un algorithme de *clustering*. C'est une méthode basée sur la densité.

Algorithme *DBSCAN* :

1. Soit *minPoints* un nombre réel positif.
2. Un point P est dense seulement si | {Q/ d(P,Q) <= *Epsilon*}| >= *minPoints*
3. Si P et Q appartiennent au même cluster, et d(Q,R) <= *Epsilon* et Q est dense, Alors P et R appartiennent au même cluster.
4. Les points non-denses sont appelés points de « bordure ».
5. Les points en dehors des clusters sont appelés « bruits ».

Figure 89: Algorithme DBSCAN [18]

Les deux paramètres nécessaires de l'algorithme *DBSCAN* sont :

-Le rayon maximum situant les voisins : *Epsilon*

- Le nombre minimum de points dans le voisinage- *Epsilon** d'un point : minPoints

Pour former un cluster, on doit avoir un nombre de points (ou des instances) situés dans un rayon inférieur à *Epilson* qui est supérieur ou égal au nombre minimum de points (minPoints).

Un Point P est dense-accessible à partir de Q (c'est-à-dire Q est le centre de cluster) seulement si :

1/la distance entre P et Q (d(P,Q)) est inférieure ou égale au rayon maximum situant les voisins (*Epsilon*) c'est-à-dire : d(P,Q) <= *Epsilon* et P est au voisinage de Q.

2/Le nombre de points au voisinage de Q est supérieur ou égal au nombre minimum de points, c'est-à-dire | {Q/ d(P,Q) <= *Epsilon*}| >= *minPoints.*

Le processus de l'algorithme *DBSCAN* consiste à choisir un point P et récupérer tous les points denses accessibles à partir P tout en respectant les conditions cités ci-dessus. Si P est un centre alors un cluster est formé. Sinon si P est une limite (c'est-à-dire il est au voisinage d'un autre point), alors il n'y a pas de points accessibles de lui et donc on passe à un autre point. Sinon si le point P n'est pas ni centre ni au voisinage d'un autre point alors il s'agit d'un bruit.

L'algorithme *DBSCAN* répète ce processus jusqu'à l'épuisement de tous les points.

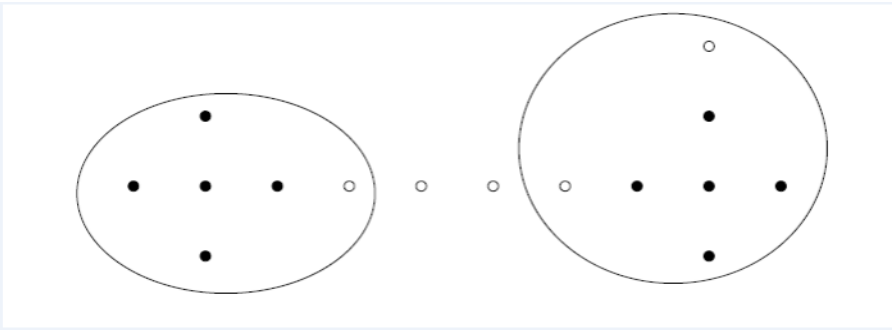

Figure 90: Clusters [18]

La figure 90 ci-dessus des clusters nous explique plus l'algorithme *DBSCAN* :

On voit deux clusters. Chaque cluster est formé en respectant les conditions de construction d'un cluster. Les points noirs, sont les points denses. Les points blancs qui appartiennent aux clusters et qui ne sont pas denses mais leur distance est inférieure ou égale à *Epsilon* sont appelés « bordures » (nous avons 3 points). Les points blancs qui sont à l'extérieur de deux clusters sont appelés « bruit » (nous avons 2 points).

Puisque l'algorithme DBSCAN est intéressant, nous l'avons appliqué sur nos données afin de construire les clusters des modèles de joueur humain et de bot. Ces modèles ont été construits en utilisant les critères suivants :

La vitesse moyenne de la trace = vitesseTot

L'accélération moyenne de la trace = Accelerations

L'écart type de la trace = Ecart Type

Les ratios de vitesse = Ratios

Les ondes moyennes = OndesMoyennes

```
Relation:     Modeles
Instances:    158
Attributes:   6
              VitesseTOt
              Accelerations
              EcartType
              Ratios
              OndesMoyennes
              Agent

DBSCAN clustering results
==========================================

Clustered DataObjects: 158
Number of attributes: 6
Epsilon: 0.9;  minPoints: 6
Number of generated clusters: 2
Elapsed time: 0, 05

(  0.) 'basse  ',0.104,0.81,1,3.06,'  Humain'          -->  0
(  1.) 'basse  ',0.109,0.96,1,3.14,'  Humain'          -->  0
(  2.) 'basse  ',0.078,1,1,3.17,'  Humain'            -->  0
(  3.) 'basse  ',0.082,0.96,1,3.23,'  Humain'          -->  0
(  4.) 'basse  ',0.094,1.04,1,3.06,'  Humain'          -->  0
(  5.) 'basse  ',0.072,1.12,1,3.33,'  Humain'          -->  0
(  6.) 'basse  ',0.08,0.94,1,3.2,'  Humain'            -->  0
(  7.) 'basse  ',0.084,0.78,1,2.96,'  Humain'          -->  0
(  8.) 'basse  ',0.095,1.12,1,3.23,'  Humain'          -->  0
(  9.) 'basse  ',0.075,1.07,1,3.21,'  Humain'          -->  0
( 10.) 'basse  ',0.155,1.17,1,3.26,'  Humain'          -->  0
( 11.) 'elevee  ',3.127,0.7,1,2.77,'  Bot'             -->  1
( 12.) 'elevee  ',3.271,0.67,1,2.7,'  Bot'             -->  1
( 13.) 'elevee  ',2.743,0.7,1,2.81,'  Bot'             -->  1
…..
( 28.) 'elevee  ',3.036,0.68,1,2.65,'  Bot'            -->  1
( 29.) 'elevee  ',2.748,0.66,1,2.89,'  Bot'            -->  1
( 30.) 'elevee  ',3.002,0.67,1,2.76,'  Bot'            -->  1
( 31.) 'elevee  ',3.024,0.63,1,2.9,'  Bot'             -->  1
( 32.) 'basse  ',0.072,0.87,1,3.1,'  Humain'           -->  0
( 33.) 'basse  ',0.085,0.98,1,3.18,'  Humain'          -->  0
( 34.) 'basse  ',0.065,1.12,1,3.25,'  Humain'          -->  0
( 35.) 'basse  ',0.25,0.87,1,3.08,'  Humain'           -->  0
( 36.) 'basse  ',0.084,0.96,1,3.15,'  Humain'          -->  0
( 37.) 'basse  ',0.083,0.81,1,3.11,'  Humain'          -->  0
( 38.) 'basse  ',0.255,1.2,1,3.22,'  Humain'           -->  0
( 39.) 'basse  ',0.078,1.02,1,3.2,'  Humain'           -->  0

Time taken to build model (full training data) : 0.05 seconds

=== Model and evaluation on training set ===
Clustered Instances

0       79 ( 50%)
1       79 ( 50%)
```

Figure 91: Application de l'algorithme DBSCAN sur nos données

La figure 91 montre l'application de l'algorithme *DBSCAN* sur nos données. Nous avons 158 point (instances). Comme paramètres à cet algorithme, on a : Epsilon= 0.9 et minPoints: 6.

Suite à nos expérimentations, deux clusters différents ont été formé. Ce sont nos modèles d'agents : Bot (50 %) et Humain (50 %).

Ces deux clusters dissimilaires qui ont des caractéristiques tellement différentes :

Cluster 0: Bot ➔ a les caractéristiques :

Vitesse Moyenne > 0.39
Accélération > 0.62
Ecart Type > 1.11
Ratios [0.82 ; 1.08]
Ondes moyennes <= 3.34

Cluster 1: Humain ➔ a les caractéristiques :

Vitesse Moyenne <= 0.39
Accélération <= 0.62
Ecart Type <= 1.11
Ratios < 0.82 et > 1.08
Ondes moyennes > 3.34

Ainsi, nos modèles d'agents sont construits : un modèle pour le joueur humain et un modèle pour le bot. Ici on ne parle pas de taux d'erreur mais plutôt de bruit et dans notre cas, on n'a pas de bruit (c'est-à-dire les données ont été bien classées et le bruit est égale à 0). En effet, chacun de nos points (instance) fait partie d'un voisinage d'un cluster.

Suite à ce travail, nous avons juste à comparer les comportements des agents à ces modèles afin de détecter les similarités entre eux et déterminer par suite s'il s'agit d'un joueur humain ou d'un bot. Cette méthode nous a permis de classifier nos agents et par la suite détecter les bots dans les jeux vidéo multi-joueurs en ligne en utilisant les techniques de forage de données.

4.13 ANALYSE DE DEPLACEMENTS DES AGENTS

Déplacer l'avatar est une activité essentielle dans les jeux vidéo multi-joueurs en ligne. En effet, ceci permet l'agent qui joue de tuer ses ennemis, de fuir ses prédateurs, de gagner l'argent… C'est pour cela nous avons décidé d'analyser la façon de se déplacer pour les deux types d'agents

(joueur humain et bot) pour voir si tous les deux, ils se déplacent de la même façon ou non. C'est pour cela nous avons travaillé sur une trace humaine, une trace de *Crbot*, une trace d'*Eraser* et une trace d'*Ice*. Ce sont quatre agents qui ont joué sur la même carte de jeu.

Pour ce fait, nous avons calculé la valeur de déplacement sur des intervalles de 200s entre le point de départ et le point d'arrivée, pour chacun de ces quatre agents. Puis nous avons fait les courbes représentatives de déplacements sur les trois axes X, Y et Z. En effet, sur la carte de jeu :

Au niveau de l'axe X : l'agent peut aller à gauche ou à droite
Au niveau de l'axe Y : l'agent peut aller en avant (avancer) ou en arrière (reculer)
Au niveau de l'axe Z : l'agent peut aller en haut (sauter) ou en bas (tomber)

Donc pour nos courbes représentatives :
S'il s'agit d'un déplacement positif sur l'intervalle de 200s, nous avons utilisé la valeur 1pour le représenter.
S'il s'agit d'un déplacement négatif sur l'intervalle de 200s, nous avons utilisé la valeur -1pour le représenter.
S'il s'agit d'un non déplacement sur l'intervalle de 200s, nous avons utilisé la valeur 0 pour le représenter.

Par exemple : Sur l'axe Y :
L'avancement est représenté par la valeur 1
Le reculement par la valeur -1
Et le non changement est par la valeur 0.

Suite à la représentation de déplacements des agents sur les trois axes X,Y et Z, nous avons obtenu,
Pour *Crbot* :

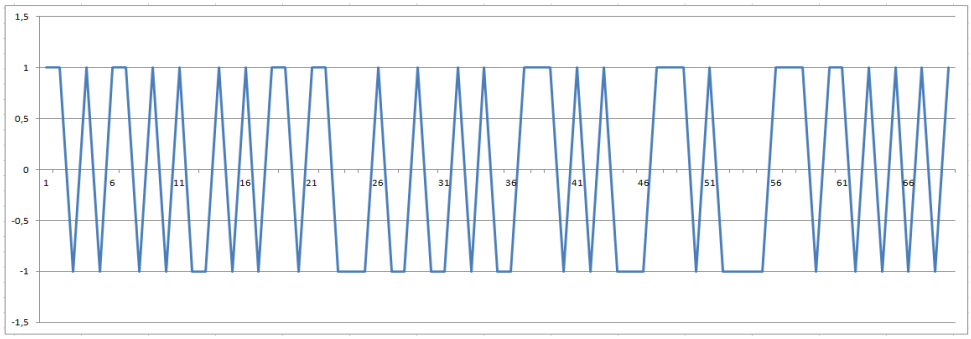

Figure 92: Déplacement sur l'axe X pour Crbot

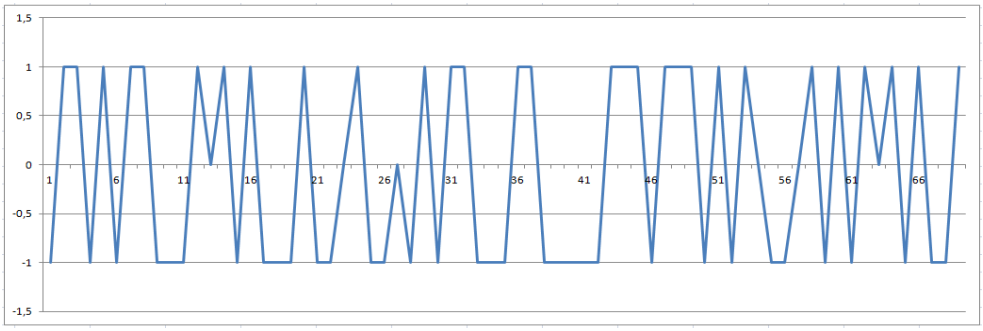

Figure 93: Déplacement sur l'axe Y pour Crbot

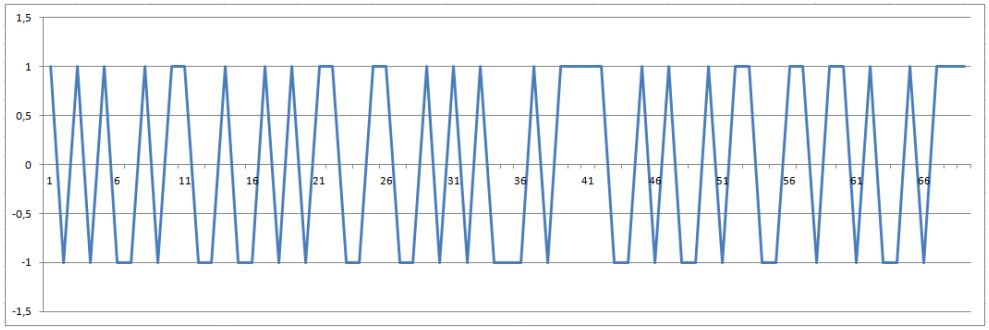

Figure 94: Déplacement sur l'axe Z pour Crbot

Pour *Eraser* :

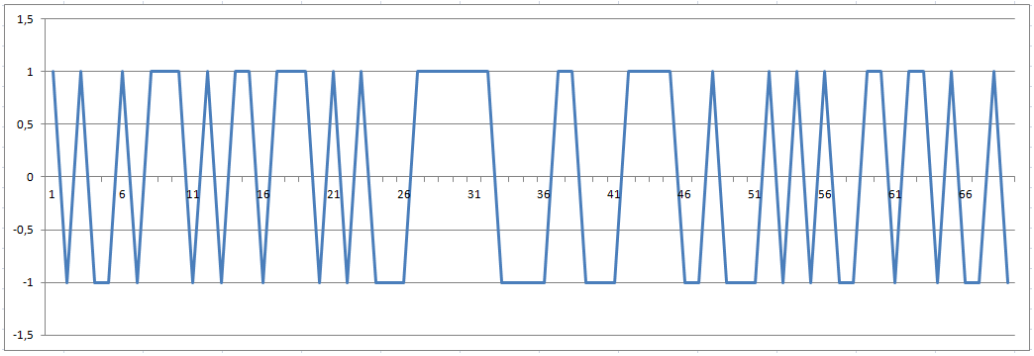

Figure 95: Déplacement sur l'axe X pour Eraser

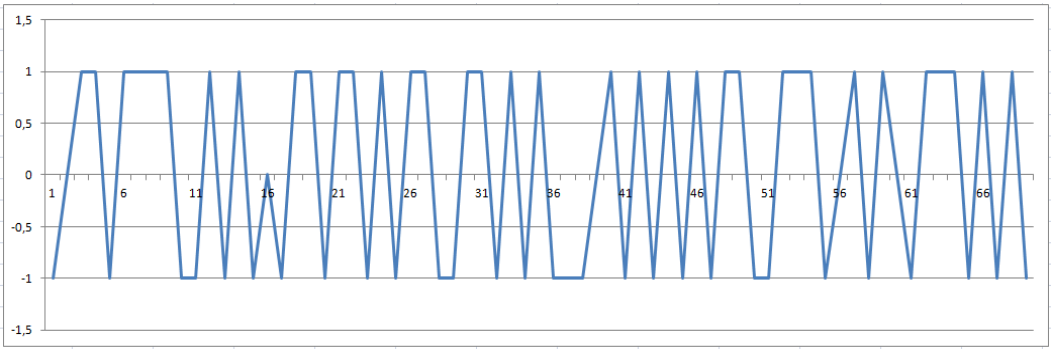

Figure 96: Déplacement sur l'axe Y pour Eraser

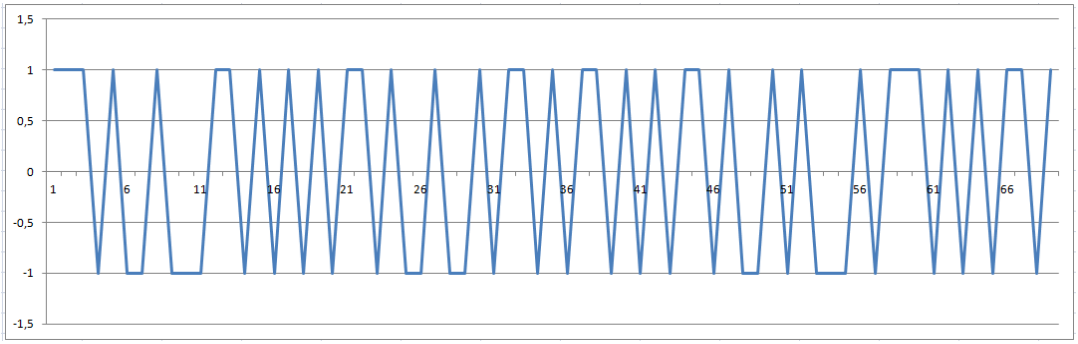

Figure 97: Déplacement sur l'axe Z sur Eraser

Pour *Ice* :

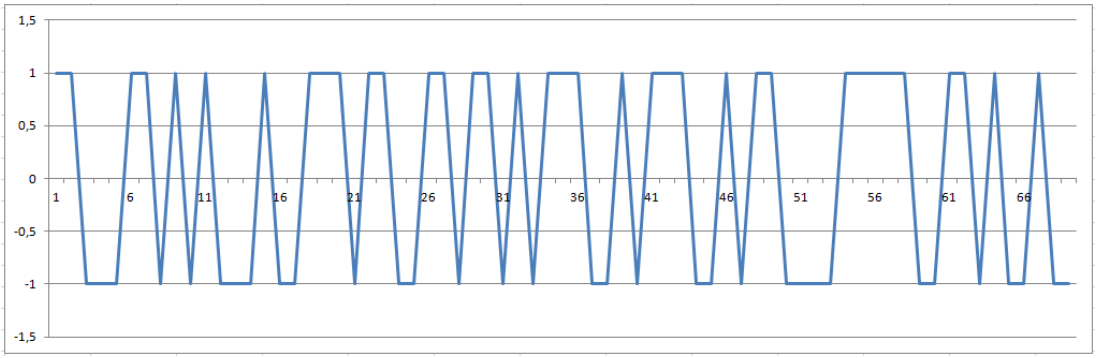

Figure 98: Déplacement sur l'axe X pour Ice

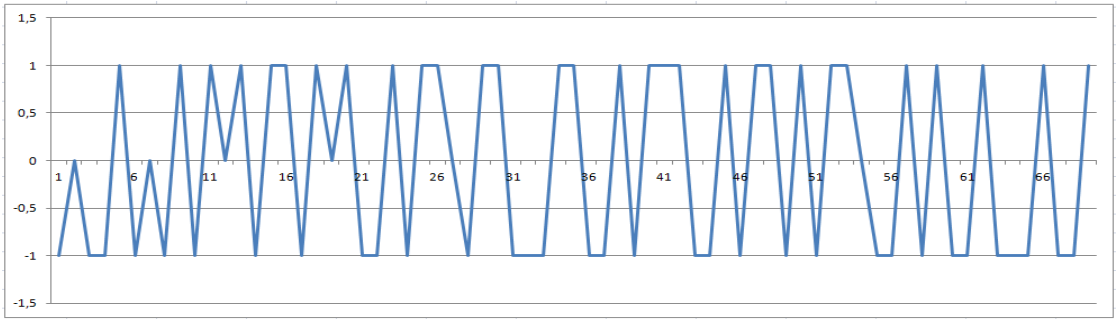

Figure 99: Déplacement sur l'axe Y pour Ice

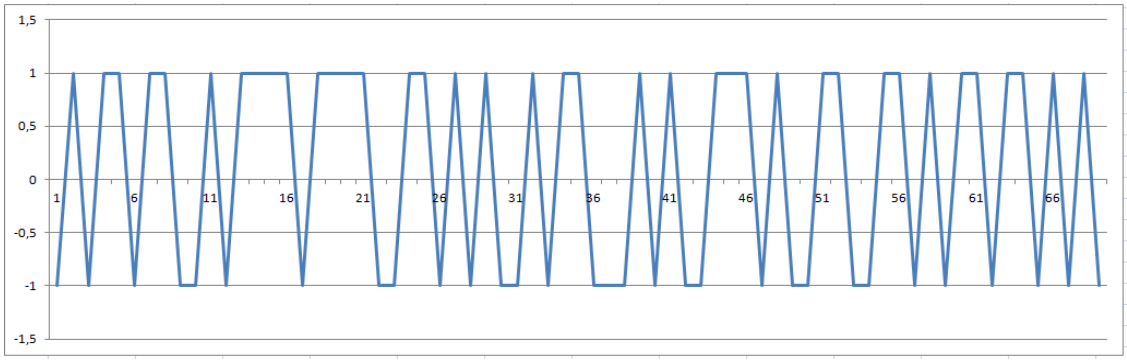

Figure 100: Déplacement sur l'axe Z pour Ice

Dans les courbes de déplacements des bots, nous avons remarqué qu'il y'en a la périodicité dans leurs déplacements (déplacement positif puis négatif puis positif…), c'est un déplacement cyclique pour les 3 bots sur les 3 axes. De plus, ils n'ont pas de périodes de repos. C'est normal puisqu'ils sont des programmes qui exécutent des algorithmes de manière cyclique.

Le déplacement de joueur humain :

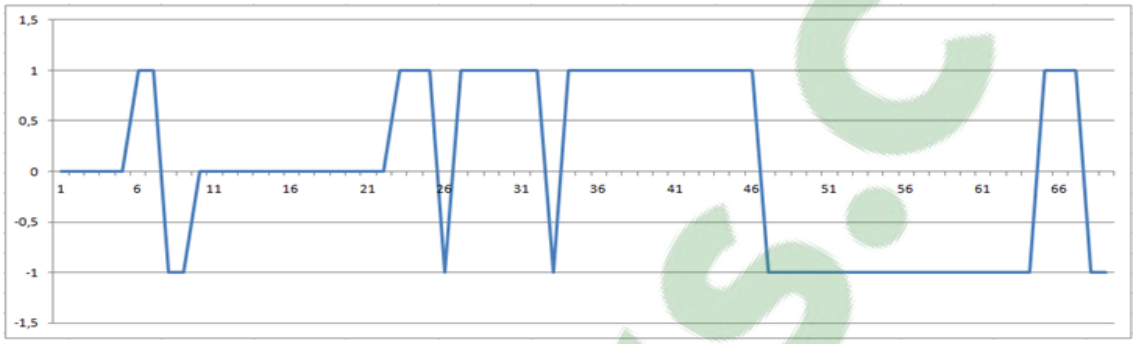

Figure 101: Déplacement sur l'axe X pour le joueur humain

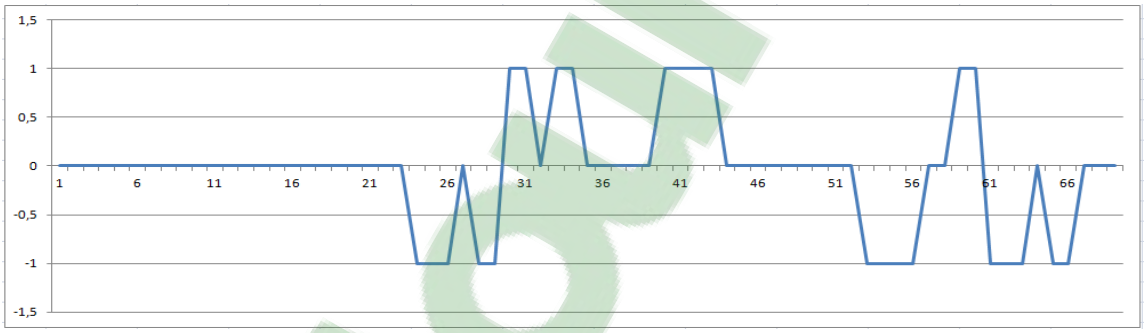

Figure 102 : Déplacement sur l'axe Y pour le joueur humain

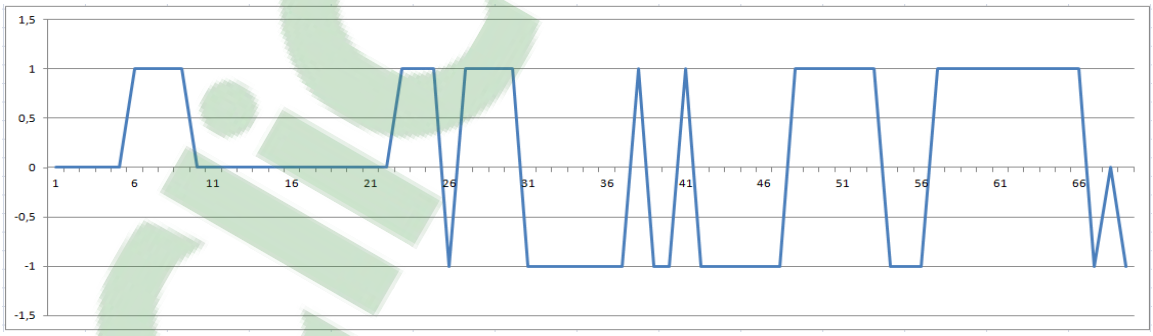

Figure 103: Déplacement sur l'axe Z pour le joueur humain

D'après les courbes de déplacements de joueur humain, nous remarquons que son déplacement n'est pas du tout cyclique, de plus il a beaucoup des périodes de repos. Aussi, le joueur humain a beaucoup de mouvements monotones, il reste sur le même sens de mouvement.

Aussi, en comparant entre les courbes de déplacements des agents, nous remarquons que les bots se déplacent simultanément, sur les 3 axes (x,y,z) d'une manière périodique. Alors que le joueur

humain se déplace à chaque fois sur un seul axe. Par exemple, il se déplace sur l'axe Y, il avance et il recule tout en gardant les mêmes coordonnées X et Z c'est pour cela on trouve sur leur représentations la valeur 0 c'est-à-dire qu'il n'ya pas de changement au niveau de coordonnées X et Z.

La manière de se déplacer diffère aussi d'un bot à un autre. Mais ce qu'ils ont en commun est que leurs déplacements sont cyclique et qu'ils n'ont pas de périodes de repos.

Comme le montre cette expérimentation, la manière de se déplacer des agents peut être aussi une façon pour faire la différence entre un joueur humain et un bot. Ce qui nous permet de détecter les bots dans les jeux vidéo multi-joueurs en ligne.

Suite à ça, nous avons passé à la classification des déplacements afin de déterminer une méthode automatique qui nous permettra de détecter les bots dans les jeux vidéo multi-joueurs en ligne.

Comme idée de classification, nous avons joint chaque trois valeurs représentatives de déplacements ensemble pour avoir la valeur finale.

Par exemple :

Si on prend 3 intervalles où un agent fait trois déplacements positifs : ça donne :

Intervalle 1 -> positif = 1, intervalle 2 -> positif =1, intervalle 3 -> positif =1

Donc l'agent obtient 1+1+1=3 pour ces 3 intervalles comme valeur finale.

Si par exemple il fait : Intervalle 1 -> négatif = -1, intervalle 2 -> positif =1, intervalle 3 -> négatif =-1

Donc il obtient -1 comme valeur finale (1-1-1=-1).

En utilisant cette méthode, nous avons calculé les valeurs finales pour les 4 agents : joueur humain, *Crbot*, *Eraser* et *Ice* sur les trois axes X, Y et Z. Nous avons joint tous les résultats pour les classifier.

La représentation de nos données de l'axe X, sous weka :

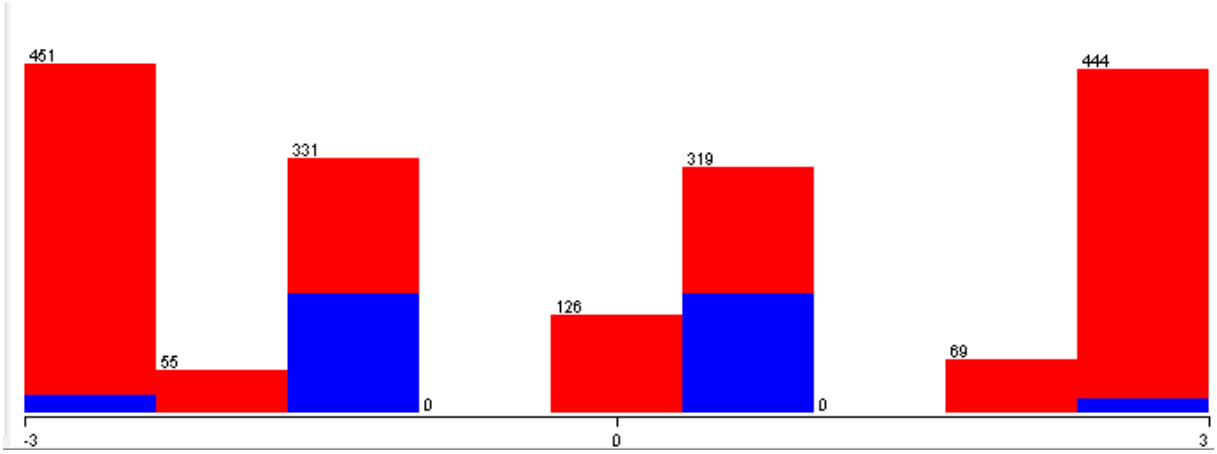

Figure 104: Répresentation de données de l'axe X sous weka

Ici nous voyons que les résultats varient entre -3 et 3.

En utilisant l'algorithme J48, nous avons classifié nos données :

```
J48 pruned tree
------------------
Time taken to build model: 0.05 seconds
=== Stratified cross-validation ===
=== Summary ===
Correctly Classified Instances        1442               80.3343 %
Incorrectly Classified Instances       353               19.6657 %
=== Confusion Matrix ===
a    b   <-- classified as
0  353 |    a =  Bot
0 1442 |    b =  Humain
```

Figure 105: Classification supervisée de données de X avec J48

Suite à la classification supervisée de données de X avec l'algorithme *J48*, nous avons obtenu 2 classes : Bot et Humain. Le taux de classification (le taux des instances correctement classées) est 80.33 % avec un taux d'erreur de 19.66 %

La représentation de données de Y :

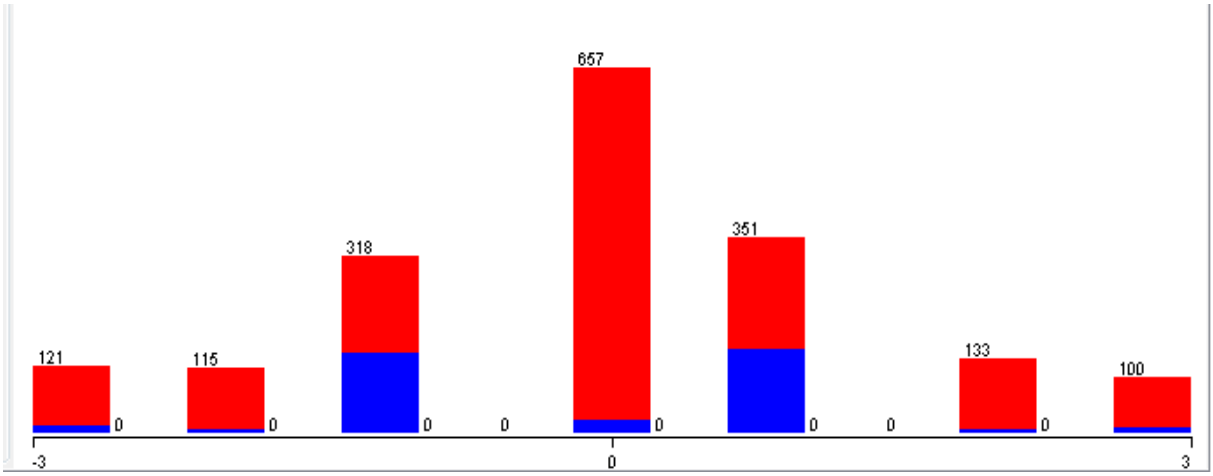

Figure 106: Représentation de données de Y sous weka

Les données de l'axe Y varient entre -3 et 3. Nous voyons 2 classes.

Comme résultat de classification supervisée de données de l'axe Y avec l'algorithme J48, nous avons obtenu 2 classes : Humain et bot. Le taux de classification est de 80.33 % avec un taux d'erreur de 19.67 %

La représentation de données de Z :

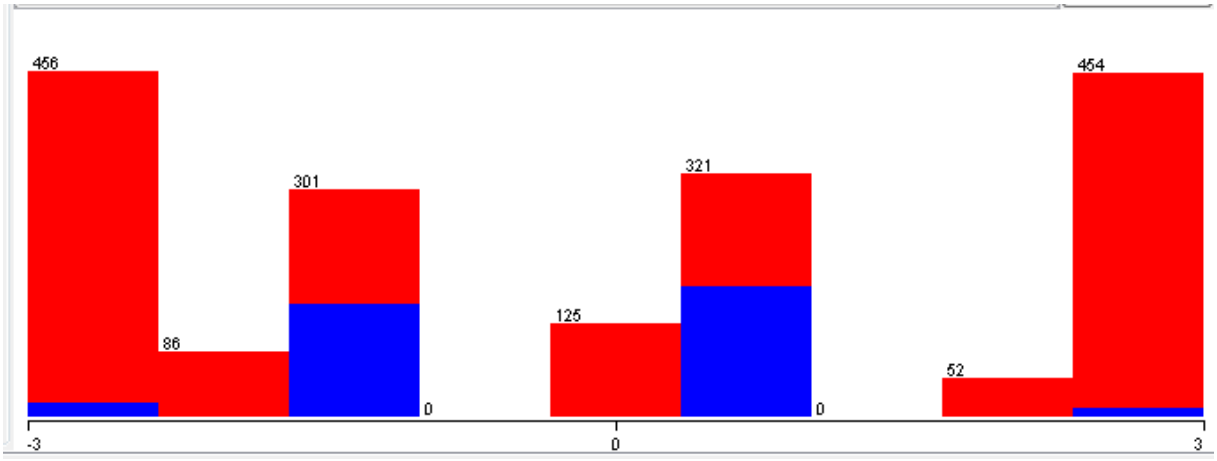

Figure 107: Représentation de données de l'axe Z sur weka

Les données de l'axe de Z varient entre -3 et 3. On voit deux classes. Comme résultat de classification avec l'algorithme j48, nous avons obtenu deux classes : Bot et Humain.

Le taux de classification est 81.11 % et le taux d'erreur est 18.86 %.

Comme le montrent les figures ci-dessus, les trois classifications pour les trois axes X, Y et Z ont presque les mêmes résultats. Ils ont tous les trois un taux de classification de 80 à 81 % et un taux d'erreur de 18 à 19 %.

Comme on a mentionné précédemment, les bots ont des mouvements cycliques. C'est-à-dire leurs déplacements doivent être:

Positif, négatif, positif ➔ comme résultat final ça donne 1-1+1=1 ou
Négatif, positif, négatif ➔ le résultat final est -1+1-1= -1

Donc au niveau de nos résultats finaux, les bots doivent avoir sur les trois axes X, Y, Z les valeurs 1 et -1. Même si parfois sur les courbes de mouvements de bots, ces derniers continuent dans la même direction sur deux intervalles successifs, c'est-à-dire ils font comme déplacements:

Positif, positif, négatif ➔ résultat final est 1+1-1= 1 ou
Négatif, négatif, positif ➔ résultat final = -1

Donc un bot possède toujours, dans tous les cas, deux possibilité de résultat final : 1 ou -1, puisque il ne fait jamais des périodes de repos.

Par contre pour le joueur humain, son mouvement n'est pas du tout cyclique, il est monotone et il continue toujours vers la même direction sur plusieurs intervalles successifs. Donc son mouvement est :

Positif, positif, positif ➔ donc il a comme valeur finale 3 ou
Négatif, négatif, négatif ➔ donc il a comme valeur finale -3

Donc comme résultat final, un joueur humain a généralement 3 ou -3. Mais nous voyons sur les courbes de déplacement de joueur humain qu'il change de direction de temps en temps. Ce n'est pas cyclique mais c'est au hasard de plus il fait beaucoup de périodes de repos, donc comme mouvement il a:

Positif, repos, repos ➔ résultat final = 1 ou
Repos, repos, positif ➔ résultat final =1 ou
Positif, négatif, négatif ➔ résultat final = -1 ou
Négatif, positif, positif ➔ résultat final = 1 ou
Repos, négatif, négatif ➔ résultat final = -2 ou
Repos, positif, positif ➔ résultat final = 2 ou
Positif, positif, repos ➔ résultat final = 2 ou
Négatif, négatif, repos ➔ résultat final = -2

Tous ces résultats possibles pour le joueur humain expliquent le taux d'erreur allant jusqu'à 19 % lors de la classification de mouvements des agents.

Donc pour conclure, l'addition de trois valeurs représentatives des déplacements des agents est de 1 ou -1 pour les bots et de -3 ou 3 pour les joueurs humains.

L'analyse de façon de se déplacer des agents est un autre critère pour faire la différence entre un joueur humain et un bot. En effet, il s'agit d'une autre manière de détection des bots dans les jeux vidéo multi-joueurs en ligne.

4.14 CONCLUSION

Nous avons proposé des nouvelles approches pour la détection des bots dans les jeux vidéo multi-joueurs en ligne. Ce sont des approches générales qui peuvent être appliquées sur n'importe quel jeu vidéo multi-joueurs en ligne et n'importe quelles traces. Nos expérimentations sur des traces réelles montrent que les joueurs humains et les bots ont des comportements très dissimilaires. La performance de nos résultats de classifications montre le succès de notre travail qui a pour objectif de détecter les bots dans les jeux vidéo multi-joueurs en ligne.

CHAPITRE 5

CONCLUSION

Au cours de notre projet de recherche, nous avons travaillé sur trois domaines importants : les jeux vidéo multi-joueurs en ligne, la sécurité de jeux vidéo et le forage de données. Chacun de ces trois domaines, est très vaste et a fait l'objet de plusieurs recherches.

Au cours des chapitres précédents, nous avons pu définir et comprendre le monde des jeux vidéo multi-joueurs en ligne et les bots des jeux qui constituent une forme de tricherie. Par la suite nous avons défini le forage de données, ses utilisations ainsi que la classification supervisée et la classification non supervisée de données.

L'intention principale était la détection automatique des bots dans les jeux vidéo multi-joueurs en ligne. Différentes approches ont été proposées pour résoudre cette problématique. Chacune de ces approches se concentre sur un seul critère pour faire la différence entre un bot et un joueur humain. En effet, chacune des solutions proposées, a été développée pour une utilisation particulière.

Notre approche consiste à travailler sur plusieurs critères pour détecter automatiquement les bots, en appliquant les techniques de forage de données et à construire un modèle complet d'agent (un modèle pour le bot et un modèle pour le joueur humain) pour pouvoir classifier les agents en comparant leurs profils à nos modèles et déterminer la similarité entre eux. En effet, en analysant les traces des jeux et en faisant les différentes expériences, nous avons réussi à répondre aux objectifs fixés pour ce mémoire.

5.1 OBJECTIFS REALISÉS

L'objectif principal de notre travail consistait à élaborer une approche qui se base sur le forage de données et qui permet de détecter automatiquement les bots dans les jeux vidéo multi-joueurs en ligne. Pour ce faire, certains objectifs intermédiaires étaient importants à réaliser.

Le premier objectif était d'acquérir des connaissances spécialisées sur le monde des jeux vidéo multi-joueurs en ligne et les bots de jeux. Notre but ici était de bien comprendre le fonctionnement des bots et d'anticiper des solutions à cette problématique.

Le deuxième chapitre a répondu à notre deuxième objectif qui proposait une étude détaillée sur le forage de données et ses techniques que nous avions l'intention de l'utiliser dans notre approche. Cette étude nous permettait d'approfondir nos connaissances sur la classification supervisée et la classification non supervisée et de mieux choisir les algorithmes que nous utilisions au cours de notre travail.

Notre troisième étape consistait à faire une investigation sur les approches existantes qui permettent la détection automatique des bots dans les jeux vidéo en ligne. Le troisième chapitre a présenté les différentes méthodes existantes pour la détection de bots dans les jeux vidéo multi-joueurs en ligne.

À la lumière de trois étapes précédentes, nous avons pu réaliser notre objectif, qui consistait à détecter automatiquement les bots dans les jeux multi-joueurs en ligne. En effet, suite à plusieurs expérimentations, nous avons pu proposer une nouvelle approche pour la détection des bots dans les jeux vidéo multi-joueurs en ligne en appliquant les approches de forage de données et nous avons pu construire un modèle complet pour chaque type d'agent (bot et humain) qui a chacun des caractéristiques bien précises (vitesse moyenne, accélération moyenne, écart type moyen, etc.)

Aussi, nous avons pu analyser les déplacements de chacun des joueurs. En effet, nous avons analysé leurs déplacements sur les trois axes : X, Y et Z ainsi que leurs façons de se déplacer. Suite à cette analyse nous avons remarqué des différences majeures entre les deux types de joueurs.

Sommairement, nous avons pu répondre aux objectifs fixés au début de notre travail et nous avons trouvé des solutions à la problématique des bots dans les jeux vidéo multi-joueurs en ligne, mais des améliorations de travail sont possibles pour donner les meilleures solutions à notre problématique.

5.2 LIMITATIONS ET AMELIORATIONS

Ce mémoire a présenté notre approche qui consiste à détecter les bots dans les jeux vidéo multi-joueurs en ligne en appliquant les techniques du forage de données. En effet, en se basant sur différentes expériences et plusieurs critères nous avons réussi à construire un modèle complet d'agent pour le joueur humain et un autre modèle pour le bot. Ainsi, nous avons réussi à déterminer la façon de se déplacer de chacun de type d'agent (joueurs humains ou bots). En plus, ce mémoire a expliqué le forage de données et a présenté ses utilisations au cours de notre travail qui nous ont aidé à faire la différence entre les joueurs humains et les bots.

Les résultats obtenus lors de nos expérimentations sont promoteurs, et notre approche repend aux objectifs fixés auparavant. Cependant, elle souffre de quelques limitations. En effet, au niveau de travail sur les longueurs d'ondes des traces, une classification des données n'était pas possible car les deux types d'agents ont des valeurs semblables et donc une détection automatique des bots dans les jeux vidéo en ligne en appliquant les techniques de forage de données et en se basant sur la longueur d'onde n'est pas possible. Il faut faire une analyse des courbes de variation des écarts types afin de déterminer s'il s'agit d'un bot ou d'un joueur humain.

Une deuxième limitation de cette approche consiste à son incapacité de déterminer le nombre des ennemis que l'agent a tué ainsi que les fois où il a été attaqué, puisque les traces sur lesquelles nous avons travaillé ne nous montrent que le déplacement de chaque agent sur les 3 axes x, y et z.

Cependant, cette technique pourrait être améliorée. En effet, en analysant plus les déplacements des agents, nous pourrons déterminer s'il existe une relation entre les déplacements des bots et ceux de joueurs humains. Aussi nous pourrons déterminer s'il existe une certaine manière de se déplacer qui se répète sur une certaine période, pour les bots, ou s'ils se déplacent spontanément. En d'autres mots, nous pourrons analyser les comportements des agents surtout ceux des bots afin de déterminer s'ils se déplacent spontanément, tout dépend de la carte de jeu ou s'ils se déplacent de la même façon sur toutes les cartes.

Aussi, nous pourrons travailler sur d'autres traces qui montrent le nombre des ennemis tués et le nombre de sessions jouées par l'agent. En utilisant ces traces, nous pourrons calculer le taux de précision (taux de précision = nombre de ennemis tués pendant une session/nombre des ennemis). De plus, nous pourrons calculer le taux de connexions en travaillant sur le nombre de sessions jouées (taux de connexions = nombre de sessions jouées pendant une certaine période/nombre de connexions moyen pendant une certaine période), puis classifier nos données selon ces deux derniers critères.

5.3 BILAN PERSONNEL DU TRAVAIL DE RECHERCHE

Pour conclure, j'aimerai profiter de ces derniers mots pour décrire mon bilan personnel du ce travail de recherche. Bien que ce projet n'était pas toujours facile, il m'a permis d'apprendre des nouvelles connaissances sur le domaine de sécurité de jeux vidéo et la programmation java et d'approfondir mes recherches sur plusieurs domaines tels que les jeux vidéo, la sécurité des jeux vidéo ainsi que le forage de données.

De plus, cette belle expérience m'a permis de faire partie de la formidable équipe de LIF (Laboratoire Informatique Formelle) et de travailler en équipe. Cette expérience enrichissante m'a permis de développer des compétences qui me seront utiles au futur.

En résumé, cette expérience positive dans le domaine de recherche scientifique s'est bien déroulée et m'a encouragé à poursuivre mes recherches et à entamer des études doctorales.

Bibliographie

1. Ruth L. Diaz, U. W. (s.d.). Violent Video Game Players and Non-Players Differ on.
2. Steven Gianvecchio, Z. W. (s.d.). Battle of Botcraft: Fighting Bots in Online Games with Human Observational Proofs.
3. Association Canadienne du Logiciel de Divertissement, 13 Novembre 2015, le secteur canadien du jeu vidéo poursuit son essort, renforçant son importance au sein de la nouvelle économie
4. Kuan-Ta Chen, A. L.-K.-H. (s.d.). Game Bot Detection Based on Avatar Trajectory.
5. Stefan Mitterhofer, C. K. (s.d.). Server-Side Bot Detection in Massive Multiplayer Online Games.
6. YEE, N. (2006). Motivations for Play in Online Games.
7. Rooij1, A. J. (s.d.). Online video game addiction: identification of addicted adolescent gamers.
8. Hall, M. (s.d.). The WEKA Data Mining Software: An Update.
9. Ayre, L. B. (2006). *Data Mining.*
10. Alexandru Iosup, Y. G. (s.d.). The Game Trace Archive.
11. Usama Fayyad, G. P.-S. (1996). *From Data Mining to Knowledge Discovery in Databases.*
12. Yeung, S. (s.d.). Detecting Cheaters for Multiplayer Games: Theory,design and Implementation
13. Ricardo Aler, J. M. (s.d.). Programming Robosoccer agents by modeling human behavior .
14. Igor V. Karpov, J. S. (s.d.). Believable Bot Navigation via Playback of Human Traces.
15. Chen, K.-T. (s.d.). Identifying MMORPG Bots: A Traffic Analysis Approach.
16. Yampolskiy, R. v. (s.d.). Embedded Noninteractive Continuous Bot Detection.
17. Philippe Golle, N. D. (n.d.). Preventing bots from playing online games.
18. A. Bouzouanne,2015, Notes de cours
19. Andreea Florentina Jonsson, 2015, Flaming as a form of toxic online disinhibition and its triggers in World of Warcraft
20. Kevin Hottot, 15/05/2015, Blizzard aurait banni 100 000 tricheurs de World of Warcraft