**TELECOM-Nancy / Loria**

14

**Game Theory Models for Trust-based Collaboration**

**[Tapez le sous-titre du document]**

**Benjamin Maurice & David Rubino**

**Encadrant : Claudia-Lavinia Ignat & Valerie Shalin**

Sommaire

[Introduction 3](#_Toc388606101)

[1. Analyse du problème 4](#_Toc388606102)

[2. Organisation du projet 6](#_Toc388606103)

[2.1. Le logiciel zTree 6](#_Toc388606104)

[2.2. Choix de développement 6](#_Toc388606105)

[3. Résolution : présentation de l’application 8](#_Toc388606106)

[3.1. Utilisation du RMI 8](#_Toc388606107)

[3.2. Fonctions du serveur 9](#_Toc388606108)

[3.3. Fonctions du client 11](#_Toc388606109)

[3.4. Interface graphique 12](#_Toc388606110)

[3.5. Sauvegarde des données 14](#_Toc388606111)

[3.6. Intérêt des différents types de jeu 15](#_Toc388606112)

[Conclusion 16](#_Toc388606113)

[Bibliographie 17](#_Toc388606114)

# Introduction

De nombreux systèmes d’Internet utilisent aujourd’hui des ressources partagées entre plusieurs utilisateurs qui peuvent être modifiées librement. Plusieurs utilisateurs ont ainsi la possibilité de collaborer sur un même projet. C’est le cas par exemple des logiciels de gestion de version comme Git ou SVN. Dans ces logiciels, les processus d’authentification sont régis par des modèles de sécurité traditionnels qui peuvent parfois être lourd à implémenter mais aussi à utiliser. L’idée pourrait être de remplacer ces modèles de sécurité par un autre modèle basé sur la confiance. Pour tester si un tel modèle est réalisable, il faut d’abord le tester sur des utilisateurs.

Lors de ce PIDR, nous nous sommes intéressés à la réaction d’utilisateurs sur un tel système. Plus particulièrement, nous nous sommes basés sur une expérience de Joyce Berg, réalisée à l’Université d’Iowa, qui a permis d’étudier la réaction sur un groupe d’utilisateurs à qui on avait donné pour mission de collaborer lors d’échanges d’argent. Les utilisateurs pouvaient au choix décider de collaborer ou bien garder l’argent pour eux.

La première phase d’approche consistait à utiliser un logiciel existant dans le but d’implémenter divers jeux basés sur la confiance. Dans la suite de notre projet, nous avons implémenté un logiciel client-serveur permettant de reproduire l’expérience de Berg sur plusieurs individus connectés au même moment sur Internet. L’objectif est d’étudier leur réaction et, à partir des statistiques recueillies, de pouvoir étudier les comportements psychologiques des individus sur les raisons qui les ont poussé à collaborer ou au contraire à adopter une attitude plutôt méfiante. Les résultats de cette expérience peuvent ensuite être étendus au cas de logiciels d’accès concurrent comme Git ou, selon l’indice de confiance d’une personne, un utilisateur voudra collaborer ou pas avec une personne sur un même projet.

Dans un premier temps, nous analyserons le problème et décrirons l’expérience de Berg. Dans un second temps, nous parlerons de l’organisation de notre projet et des logiciels que nous avons été amenés à utiliser et implémenter. Enfin, nous détaillerons l’architecture de notre logiciel ainsi que ses différents fonctionnalités.

# Analyse du problème

Actuellement, les systèmes distribués facilitent la collaboration entre plusieurs utilisateurs qui collaborent sur un même réseau ou projet, sans contrôle nécessaire par une autorité supérieure telle un super administrateur. Parmi ces systèmes, on trouve notamment les réseaux peer-to-peer ou cloud. Ces systèmes sont peu couteux à mettre en place et permettent la mise en relation de milliers d’utilisateurs qui peuvent se composer en groupe. Cependant, de tels systèmes posent de nouveaux problèmes de sécurité concernant les droits d’accès aux fichiers. Il faut s’assurer que l’utilisateur possède les droits correspondant selon son statu avant de pouvoir lui fournir un accès au fichier. De même, il est important de savoir si l’utilisateur a des droits de lecture ou écriture sur le fichier. Or, les modèles classiques de sécurité ne sont pas très bien adaptés pour la gestion de tels systèmes.

L’une des solutions investiguées par l’équipe SCORE est de remplacer ces modèles de sécurité par d’autres moins restrictifs pour les utilisateurs (TRUONG, IGNAT, & MOLLI, 2012). L’idée est d’abord de donner accès aux données à un utilisateur, puis de vérifier si les conditions de sécurité ont bien été respectées après. Si tel est le cas, l’utilisateur ne sera pas restreint sur son accès aux données. Dans le cas contraire, des restrictions particulières pourraient être mises en place sur cet utilisateur spécifique.

Les avantages d’un tel système sont nombreux. Tout d’abord, le processus de sécurité est allégé, ce qui simplifie la tache pour l’administrateur. Ensuite, du coté utilisateur, le fait qu’il n’y ait pas de vérification pour accéder aux données rend la volonté de partager et accéder aux données plus facile. Un processus lourd peut décourager plus facilement certains utilisateurs qui, au final, avaient une volonté de collaborer sur un projet. Ainsi, de telles améliorations entrainent un gain de temps et d’argent considérables, qui sont des aspects à ne pas négliger lors de la mise en place d’un système.

Le meilleur moyen de tester ce système est de faire une simulation avec des utilisateurs sélectionnés pour l’occasion. Une telle expérience a été conduite par Joyce Berg pour l’Université d’Iowa (BERG, 1994). Dans cette expérience, plusieurs sujets étaient sélectionnés et disposés dans deux salles différentes (Room A et B). Une troisième salle (Room C) contient des enveloppes. Chaque utilisateur dispose à la base de $10. Aléatoirement, à chaque utilisateur de Room A est associé un utilisateur de Room B. Le but de l’expérience était, pour chaque paire d’utilisateur, de s’échanger de l’argent par l’intermédiaire des enveloppes contenues dans Room C. L’utilisateur a soit le choix de collaborer et de donner ses $10 à son partenaire, soit de garder l’argent pour lui. Dans le cas où il y a collaboration, la somme reçue par le deuxième joueur est le triple de la somme donnée par le premier. Ce deuxième utilisateur a ensuite soit le choix de garder l’argent pour lui, soit de le renvoyer – en totalité ou en partie – à son partenaire de jeu. Cependant, une condition est imposée : on ne peut renvoyer qu’au maximum le montant que l’autre utilisateur a envoyé. Le schéma ci-dessous détaille une telle simulation pour 2 échanges entre 2 utilisateurs.

**Etape 1**

Envoi de $10

**Utilisateur 2**

Somme initiale: $10

**Utilisateur 1**

Somme initiale: $10

**Etape 2**

Envoi de $10

**Utilisateur 2**

Nouvelle somme: $40 = $10 de début + 3 \* $10 envoyés

**Utilisateur 1**

Nouvelle somme: $0

**Etape 3 : fin du jeu**

**Utilisateur 2**

Nouvelle somme: $30

**Utilisateur 1**

Nouvelle somme: $10

On remarque que, dans une telle situation, si le processus de collaboration est respecté, l’utilisateur 1 ne perd pas d’argent et l’utilisateur 2 en gagne. La confiance est donc établie entre les 2 utilisateurs et, lors d’un prochain échange, les utilisateurs pourront inverser leurs rôles afin que l’utilisateur 1 puisse aussi gagner de l’argent.

Si l’expérience est extrapolée dans le contexte d’un système de gestion de fichiers, la situation est similaire. Un utilisateur ne perd rien en partageant un fichier avec un autre utilisateur. Ce dernier, en revanche, peut librement bénéficier des ressources mises à disposition et sera donc certainement très enclin à partager à son tour un autre fichier avec son collaborateur.

Ce sont de telles expériences que nous avons été amenés à simuler dans notre projet. A partir des résultats expérimentaux de Berg, nous avons décidé d’implémenter un logiciel client-serveur qui permettrait de reproduire l’expérience de Berg pour différents utilisateurs situés à des endroits divers (France, USA…).

# Organisation du projet

## Le logiciel zTree

Dans un premier temps, nous nous sommes intéressés à un logiciel qui existait déjà, et qui avait été développé par l’Université de Zurich (University of Zurich), qui permet de développer des expériences d’ordre économique. L’expérience du Trust Game pouvant aussi bien s’appliquer à l’économie, il était possible d’implémenter l’expérience sur ce logiciel. Après demande d’accord à l’université pour utiliser leur logiciel – il est nécessaire d’avoir une licence pour le télécharger – nous avons cherché à implémenter la solution. Cependant, ayant été confronté a certains problèmes – interface peu intuitive, design moyen, nombreuses options permettant difficilement de si retrouver – nous avons choisi de créer notre propre logiciel. Cela nous permettrait d’avoir plus de contrôle sur les différentes options que nous souhaitons avoir, et d’être plus facile d’utilisation.

## Choix de développement

Nous avons choisi de développer le logiciel en langage JAVA, en utilisant JavaFX pour l’interface graphique. Notre choix s’appuie sur le fait que JAVA est un langage objet, et offre donc de nombreuses possibilités grâce à l’héritage et à la généricité. De plus, il s’agit d’un langage portable, permettant ainsi d’être utilisé sous Windows, Mac OSX ou Linux, sans avoir à se soucier de problèmes de compatibilité – chose non possible en C.

Nous avons décidé d’utiliser JavaFX plutôt que les librairies Swing ou Awt de Java car le rendu est meilleur visuellement. De plus, cela nous a appris à nous servir d’une autre librairie pour l’interface graphique. Le développement de l’interface s’est fait à l’aide de JavaFX Scene Builder, qui nous permet de positionner directement les composants graphiques sur notre interface – fonctionnement similaire a celui de Visual Studio pour C#. Cela nous a permis de gagner beaucoup de temps lors du développement de l’interface.

Nous avons utilisé Eclipse comme IDE, car il donne accès à de nombreuses fonctions facilitant la tache du développeur. De plus, à l’aide d’un plugin, il est également possible de générer l’UML directement.

La méthode de développement utilisé a été une méthode agile, ce qui a permis d’implémenter progressivement l’application en lui ajoutant à chaque fois de nouvelles fonctionnalités. Nous discuterons du produit final et non pas des différentes versions dans la suite de ce rapport.

Apres étude des besoins pour réaliser l’expérience, nous avons relevé les points suivants que notre logiciel devait implémenter :

* Fournir une interface client-serveur permettant de lancer plusieurs clients à la fois
* Permettre à l’administrateur de régler les différents paramètres d’un jeu en fonction de l’expérience qu’il veut conduire
* Fournir à l’utilisateur des statistiques sur ses manches précédentes à l’aide d’un histogramme
* Permettre une sauvegarde des données afin d’analyser le comportement psychologique des joueurs
* Assurer le contrôle des données rentrées par l’administrateur et les joueurs

# Résolution : présentation de l’application

Notre logiciel est décomposé en 3 parties principales : une partie serveur, une partie client et une partie commune – utilisée notamment pour tout ce qui concerne la sauvegarde de données.

## Utilisation du RMI

Pour réaliser la connexion client-serveur, nous avons implémenté l’interface RMI fournie par JAVA et qui permet d’interagir entre un serveur et un client sans forcément passer par Internet. Ainsi, sur une seule machine, nous avons pu simuler les comportements des clients et du serveur simultanément sans jamais nécessiter une connexion à Internet. Le schéma ci-dessous montre le fonctionnement du RMI.

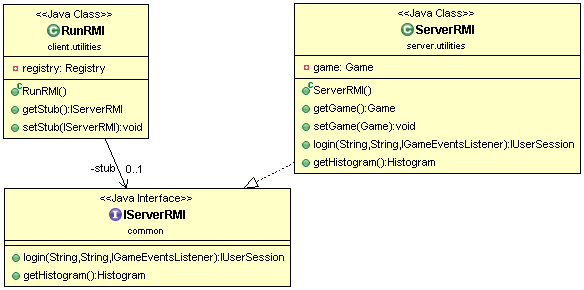


Puisque le client et le serveur vont tous les deux utiliser le RMI, nous avons créé une interface IServerRMI utilisable par les deux dans le package common. Cette interface permet de gérer l’identification à l’aide d’une méthode login et de récupérer l’histogramme correspondant a une partie.

Du coté serveur, celle-ci est implémentée dans le package server.utilities. Le serveur y gère l’authentification et permet de récupérer l’histogramme.

Du coté client, l’interface est implémentée dans la classe RunRMI dans le package client.utilities. Elle permet au client de se connecter au réseau de clients existant déjà en passant par le Stub.

L’UML ci-dessous montre la relation entres ces classes. On y retrouve le schéma général de fonctionnement ci-dessus.



## Fonctions du serveur

Le serveur doit définir les classes nécessaires pour la connexion et l’identification future des clients, ainsi que les structures correspondant au jeu de l’expérience de Berg. Ces classes sont implémentées dans le package server.model. Un utilisateur est défini par son adresse email – qui servira de login car unique – ainsi qu’un mot de passe et une somme initiale d’argent avec laquelle il commence la première manche. Cette somme pourra varier suivant la configuration du jeu choisie.

Chaque utilisateur qui arrive sur le jeu est à tour de rôle affecté à un partenaire de jeu. S’il arrive qu’il y ait un nombre impair d’utilisateur, on applique l’algorithme FIFO : le premier joueur arrivé laissera sa place au dernier arrivé des qu’il choisit d’arrêter le jeu.

La classe ***Game.java*** définit les paramètres correspondant à un jeu. Il est important de différencier un jeu d’une manche. Un jeu consiste à un jeu de paramètres déclarés par le serveur, qui peut comprendre une ou plusieurs manches. Le jeu s’arrête lorsque toutes les manches ont été jouées ou bien si tous les utilisateurs arrêtent de jouer. Une manche consiste en un jeu de répétitions paramétré par le serveur. Lorsque ce nombre de répétitions est atteint, la manche est terminée mais le jeu continue jusqu'à ce que les utilisateurs ou l’administrateur choisissent d’arrêter. Un jeu peut n’avoir qu’une manche. Un jeu est donc défini par un nombre d’utilisateurs, un nombre de coups autorisés – correspondant aux nombre de répétitions possibles – un montant de départ pour chaque utilisateur, le choix du type de jeu, un nombre de joueurs contrôlés par l’IA, et un type d’IA. En effet, plusieurs types de jeu peuvent être testés. Il est également possible de jouer tout seul contre l’ordinateur au moyen d’une intelligence artificielle.

L’état du jeu est sauvegardé dans un fichier binaire à chaque fin de jeu et chargé en mémoire à chaque début de jeu à l’aide des méthodes save() et load().

**public** **void** save() **throws** IOException {

String fileName = "game.bin";

ObjectOutputStream oos = **new** ObjectOutputStream(**new** FileOutputStream(

fileName));

**try** {

oos.writeObject(**this**);

} **catch** (IOException ioe) {

System.*err*.println("FATAL ERROR -- " + ioe.toString());

}

oos.close();

}

**public** Game load() **throws** IOException {

String fileName = "game.bin";

Game tmp = **null**;

ObjectInputStream ois = **new** ObjectInputStream(**new** FileInputStream(

fileName));

**try** {

tmp = (Game) ois.readObject();

} **catch** (IOException ioe) {

System.*err*.println("ERROR -- " + ioe.toString());

} **catch** (ClassNotFoundException cnfe) {

System.*err*.println("ERROR 'Unknown class' -- " + cnfe.toString());

}

ois.close();

**return** tmp;

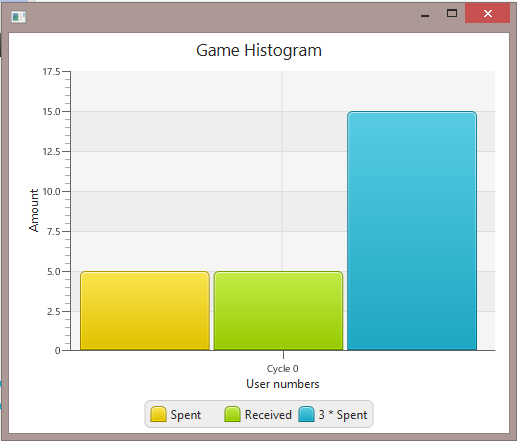
}

La classe contient également les différentes méthodes nécessaires pour retrouver un utilisateur dans une session de jeu en fonction de son login et pour l’ajouter à une session courante.

## Fonctions du client

Les classes implémentant le comportement du client se trouvent dans le package client.application. Tout d’abord, les données d’un client sont sauvegardées dans un objet ClientData qui contient le montant joué par un client lors d’une manche, le temps qu’il met à jouer, son solde courant et son adresse email. Cette classe est utilisée lors de la sauvegarde des données.

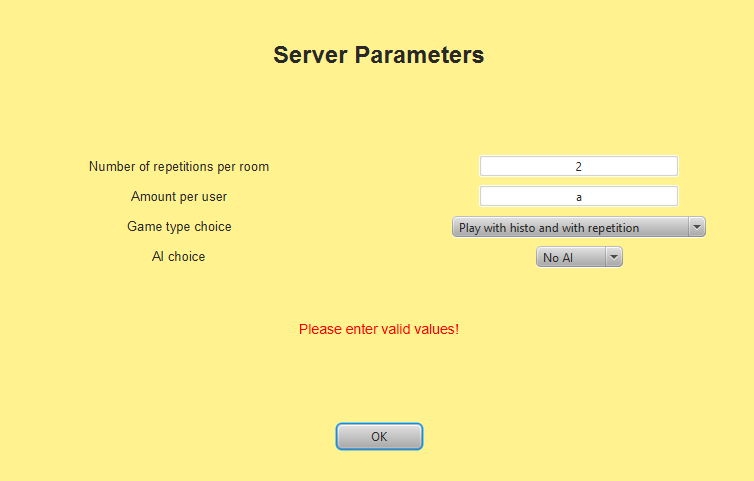
Lors du jeu, les données du client sont passées au serveur qui construit un objet DataSample a partir de ces données. La classe ***DataSample.java*** – package common – permet de définir ces données. Elles sont le support utilisé pour construire l’histogramme à l’aide de la classe ***Histogram.java*** du package common. Pour construire l’histogramme, nous nous sommes appuyés sur la classe TreeSet définie par JAVA. Il est possible de réécrire la méthode compare de cette classe afin de pouvoir comparer deux DataSet entre eux. Le client a accès directement à ces données durant le jeu. La classe ***HistogramView.java*** permet une représentation de l’histogramme. Il est important de noter que, selon le type de jeu choisi, l’histogramme ne sera pas toujours visible. Ci-dessous se trouve un exemple d’histogramme obtenu lors d’une partie.



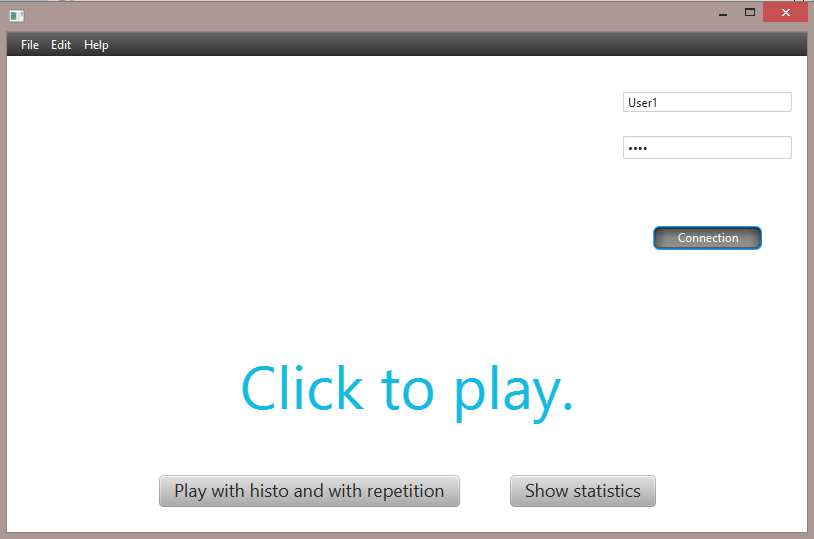
## Interface graphique

Comme spécifié plus haut, l’interface graphique a été réalisée à l’aide de JavaFX. Le code pour l’interface a été généré automatiquement par JavaFX Scene Builder après positionnement des composants sous la forme de fichier .fxml. A chaque fichier .fxml est associé un fichier .java pour le Controller. Ainsi, FrameServerController du package server.application défini l’interface graphique pour le serveur.

Le Controller effectue une vérification des données par l’administrateur lorsqu’on clique sur OK. Au cas où l’une de ces données soit invalide, on affiche un message d’erreur. Ce contrôle s’effectue dans la méthode OKIssueFired, appelée lorsqu’on clique sur OK.

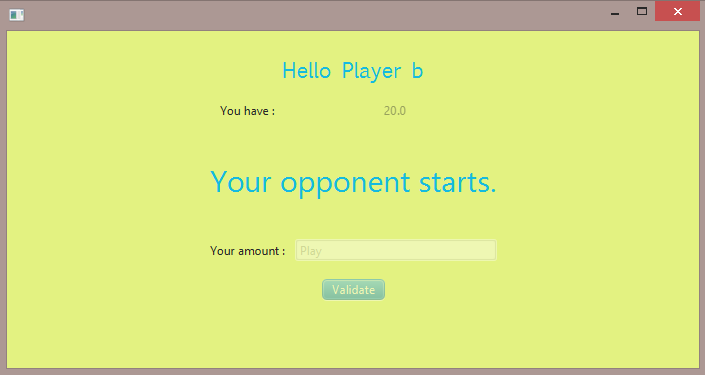


Le client possède 2 interfaces graphiques, une pour l’authentification et une pour le jeu. ***MainFrameClientController.java*** définit l’interface d’authentification, la première sur laquelle le client arrive lorsqu’il lance l’application.



Après authentification, le client peut soit jouer – il faut noter que le type de jeu sélectionné par le serveur est le seul disponible – ou bien regarder ses statistiques de jeu qui le redirigeront vers l’histogramme. Lors de l’appel sur Play, le jeu se met en attente d’autres joueurs avant de pouvoir commencer le jeu.

Lorsqu’un autre joueur entre en jeu, la fenêtre de jeu du client s’affiche. Celle-ci est définie dans ***PlayClientController.java***. Cette fenêtre affiche le solde courant du joueur et lui permet de rentrer une somme d’argent à jouer. La méthode validateIssueFired permet de vérifier si le montant entré par le joueur est correct ou non.



## Sauvegarde des données

L’objectif de notre logiciel étant d’analyser les données des utilisateurs pour pouvoir faire des statistiques par la suite, il est donc important de sauvegarder les données du client, à savoir celles trouvées dans la classe ***DataSample.java*** décrite plus haut. Les données sont sauvegardées dans des fichiers .csv. La classe ***FileSaver.java*** du package common fournit les méthodes nécessaires à la sauvegarde. Les données du server et du client sont sauvegardées dans des fichiers séparés, la date servant à l’identification d’un jeu spécifique – date sous le format MM/DD/YYYY hh:mm.

Les méthodes generateCsvServer() et generateCsvClient() permettent de créer le fichier .csv avec des en-tête de colonne. Les méthodes writeCsvClient() et writeCsvServer() écrivent les données dans la fichier à la suite des données déjà existantes. La méthode writeCsvServer() est donnée ci-après – les autres méthodes de la classe étant basées sur la même architecture.

**public** **void** writeCsvServer() {

**try** {

FileWriter writer = **new** FileWriter(*settingsServer*.getName(), **true**);

DateFormat dateFormat = **new** SimpleDateFormat("MM/dd/yyyy hh:mm a");

Date date = **new** Date();

writer.append(dateFormat.format(date));

writer.append(',');

writer.append(**this**.game.getNbRepetitions() + "");

writer.append(',');

writer.append(**this**.game.getUsersPerRoom() + "");

writer.append(',');

writer.append(**this**.game.getAmountPerUser() + "");

writer.append(',');

writer.append(**this**.game.getGameChoice() + "");

writer.append(',');

writer.append(**this**.game.getTypeAI().getModelAI());

writer.append('\n');

writer.flush();

writer.close();

} **catch** (IOException e) {

e.printStackTrace();

}

}

## Intérêt des différents types de jeu

Quatre types de jeu sont proposés du coté serveur – 5 en comptant le jeu avec intelligence artificielle:

* Jouer avec répétition et avec histogramme
* Jouer avec répétition et sans histogramme
* Jouer sans répétition et avec histogramme
* Jouer sans répétition et sans histogramme

L’intérêt de ces différents types de jeu est de tester la réaction des utilisateurs dans chacun des cas. La réaction d’un utilisateur ne va pas forcément être la même s’il joue de manière prolongé avec son partenaire ou s’il a accès aux histogrammes des parties précédentes.

Lorsqu’on joue avec répétition, il est possible, à la fin de la partie, de rejouer avec un autre utilisateur. En fonction de la partie actuelle, cela peut être déterminant pour comprendre le comportement de l’utilisateur – collaboration ou refus. De même, si les joueurs ont accès aux histogrammes des parties précédentes, ils peuvent anticiper le comportement de l’utilisateur avec lequel ils sont en train de jouer et, par conséquent, adapter leur comportement en fonction de la partie. Ainsi, les résultats obtenus pour un même utilisateur peuvent être très différent selon le type de jeu.

Il est également possible de retrouver ces configurations avec intelligence artificielle. L’administrateur peut ainsi définir plusieurs joueurs IA. Leur comportement se base sur les statistiques prélevées par les autres utilisateurs. Dans le cas de l’utilisation d’un joueur IA, on cible l’attention sur un utilisateur spécifique et comment il réagirait dans un cas plus général.

# Conclusion

Ce PIDR nous a permis d’implanter notre propre logiciel afin de reproduire l’expérience de Berg sur des utilisateurs connectés à travers Internet. A partir des données recueillies, il devient possible d’étudier le comportement des utilisateurs et de générer des modèles statistiques concernant leur comportement. Ces résultats peuvent ensuite permettre d’établir des indices de confiance qui évolueraient en fonction de la volonté des utilisateurs à collaborer entre eux plutôt qu’à rechercher un gain personnel.

De plus, l’intuitivité de notre logiciel fait qu’il est facilement utilisable et qu’il est possible de rajouter des fonctions et de poursuivre son développement. Il serait possible de rajouter une option pour permettre à plusieurs utilisateurs de former une équipe et de collaborer avec une autre équipe dans un but commun.

Un autre développement possible de ce projet pourrait consister à établir des algorithmes permettant de déterminer l’indice de confiance pour ensuite le déployer sur Internet et le tester sur de véritables utilisateurs.

Si les tests sur les utilisateurs se montrent concluants, un modèle de collaboration sur la confiance pourrait être une véritable alternative aux protocoles de sécurité traditionnels et permettre d’être implanter sur des systèmes de gestion de fichiers, comme Git ou Svn.

# Bibliographie

BERG, J. (1994). Trust, Reciprocity, and Social History. *Games and Economic Behavior 10, 122-142 (1995).* Iowa City.

TRUONG, H. T., IGNAT, C.-L., & MOLLI, P. (2012). A Contract-Extended Push-Pull-Clone Model for Multi-Synchronous Collaboration. *International Journal of Cooperative Information Systems* .

University of Zurich, D. o. (s.d.). *About z-Tree*. Consulté le Mai 17, 2014, sur IEW Zurich: http://www.iew.uzh.ch/ztree/index.php