Plateforme de Jeux en réseau

Jean Avril

February 2, 2015

Contents

1 Requètes MySQL en Php

1.1 SQL

La table utilisée est une table SQL à trois champs définis comme suit:
(string)name | (string)ip | (string)port

1.2 Requètes Disponibles

connection.php Ajoute le client à la liste et renvoi la liste des autres clients connectés (seulement les pseudos)
deconnection.php Retire le client de la liste
getList.php Renvoie la liste des clients (seulement les pseudos)
getMember.php Renvoie l'IP et le port utilisés par un client donné

1.3 Dans un explorateur web

projetcitron.fr/Game/connection.php?name=TON_PSEUDO
projetcitron.fr/Game/deconnection.php?name=TON_PSEUDO
projetcitron.fr/Game/getList.php
projetcitron.fr/Game/getMember.php?name=LE_PSEUDO_CIBLE

Attention: La casse (majuscules/minuscules) compte dans les url.

1.4 Exercice

1)regarder qui est connecté
2)se connecter
3)vérifier si tu on est dans la liste
4)récuperer l'IP et le port d'un des clients connectés
5)se déconnecter
6)vérifier qu'on n'apparait plus dans la liste

1.5 Suite

L'objectif est de récuperer l'adresse ip d'une personne connectée, puis d'établir une connection directe avec elle par socket Java en se passant du serveur.

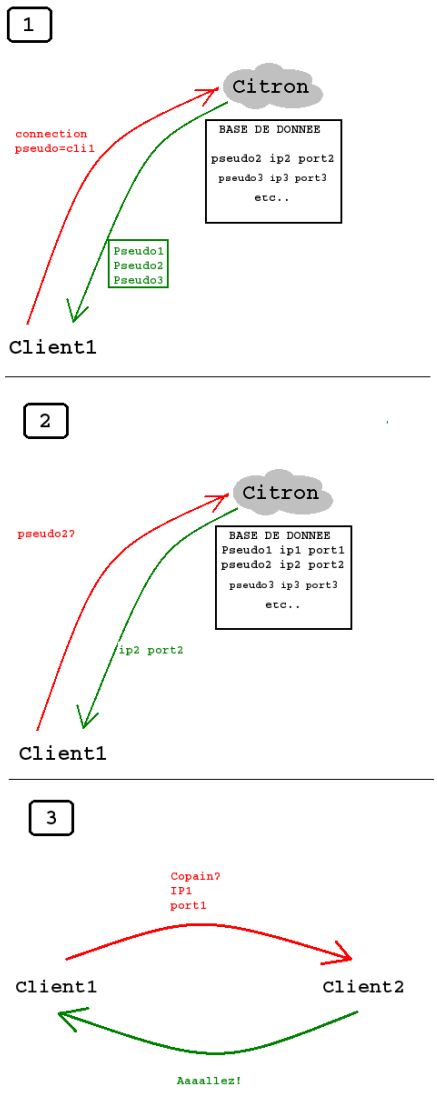

Figure 1: Illustration de l'idée de base

2 Changement: Java et PostgreSQL

Grand changement de direction:
En effet, sécurité oblige, il est très compliqué d'envoyer une "socket" directement d'un ordinateur à un autre ordinateur connecté derrière une box. En effet, il y a tout un tas de pare-feux etc. On se replie donc vers la solution utilisée par le plus grand nombre:
Les clients sont tous connectés à un serveur, qui transmettra les informations. Donc, un client n'a plus besoin de connaitre adresse ip et port de copain. Donc, il n'est pas utile de passer par des script php. (du moins, pas maintenant). On implémente donc un programme qui interagit directement avec la base de donnée.

La base de donnée a changé: au revoir les ip et ports:

NAME | DATE

De plus, la table n'est plus mysql mais postgresql.
"nom" est toujours donné par le client, avec la même condition: Si le pseudo existe déja il faut en trouver un autre. Mais "date" est plus intéressant:
Il s'agit du nombre de secondes qui se sont écoulées entre le premier janvier 1970 à minuit et maintenant. Concrêtement, ça permet de savoir depuis combien de temps on a pas eu de nouvelles d'un copain:
La fonction refresh rafraichi non seulement le client, mais aussi la base de donnée: elle note l'heure actuelle, puis pour chaque copains enregistrés, elle fait:
$heure\_maintenant$ - $derniere\_heure\_notée\_par\_copain$
Si copain n'a pas noté la bonne heure depuis plus de X secondes (on a pris 60 en experimentation), alors copain est exclu. C'est d'ailleurs pour ne pas être exclu qu'on a renoté l'heure actuelle.
Pour l'instant, refresh affiche la liste des copains encore présents, mais plus tard, elle se contentera de la noter discrètement.
L'idée est d'appeller refresh toutes les 30 secondes, pour:

1) Ne pas se faire dégager par un copain

2) Obtenir une liste actualisée de tout les copains.

3 Structure du programme

3.1 Base de données

3.1.1 Table "users"

(string)name | (long)date | (string[])friends | (string)game | (int)id
-name: Nom de l'utilisateur
-date: Date en secondes depuis le 1er janvier 1970
-friends: Liste d'amis proposant de jouer
-game: Prend le nom du jeu choisi. Exemple: "revenge"
-id: Peut prendre les valeurs "abort"(-2), "waitAnswer"(-1), ou l'identifiant d'une partie.

3.1.2 Table "revenge"

(int)id | (boolean)token | (int)x | (int)y
-id: Identifiant de la partie au sein d'un jeu, auto incrémenté.
-token: Jeton donnant le droit de jouer, initialement à "master"
-x: Abscisse du coup joué, initialement à "-1"
-y: Ordonnée du coup joué, initialement à "-1"

3.2 Menu principal

Le menu présente la liste des utilisateurs connectés.
Cette liste est rafraichie toutes les 30 secondes.
En sélectionnant un utilisateur, on peut choisir dans une liste un jeu auquel jouer avec lui.

3.3 Amorçage d'une partie

3.3.1 Côté "maitre"

Le maître est celui qui propose la partie. Pour proposer une partie à l'esclave, il fait les actions suivantes:
-Il place son nom dans le champ "friends" de l'esclave.
-Il initialise son propre champ "game" à la valeur correspondant au jeu choisi.
-Il initialise son propre champ "id" à la valeur "waitAnswer"(-1).

Puis il attend une réponse, en ayant la possibilité d'abandonner.
-Si il abandonne, il initialise son propre champ "id" à la valeur "abort"(-2).
-Si son propre champ "id" prend la valeur "abort", la demande de jouer est refusée.
-Si son propre champ "id" prend une valeur différente de "waitAnswer" ou "abort", alors la demande est acceptée.

La partie commence avec les paramètres (game, id, master), ce qui signifie que le jeu joué est "game", les informations de la partie se trouvent dans le champ "id" de "game" et le rôle de l'utilisateur est "master".

3.3.2 Côté "esclave"

Pour accepter une partie proposée, l'esclave fait actions suivantes:
-Il choisi un nom dans son propre champ "friends".
-Il récupère le champ "game" associé à ce nom.
Si il accepte:
-Il créé une nouvelle partie dans le jeu "game" et il récupére l'id de cette partie.
-Il place l'id de la partie dans le champ "id" associé au friend choisi.
-Il retire le friend de son propre champ "friend".
La partie commence avec les paramètres (game, id, slave).

Si il refuse:
-Il place la valeur "abort" dans le champ "id" du friend.
-Il retire le friend de son propre champ "friend".

3.3.3 Illustration

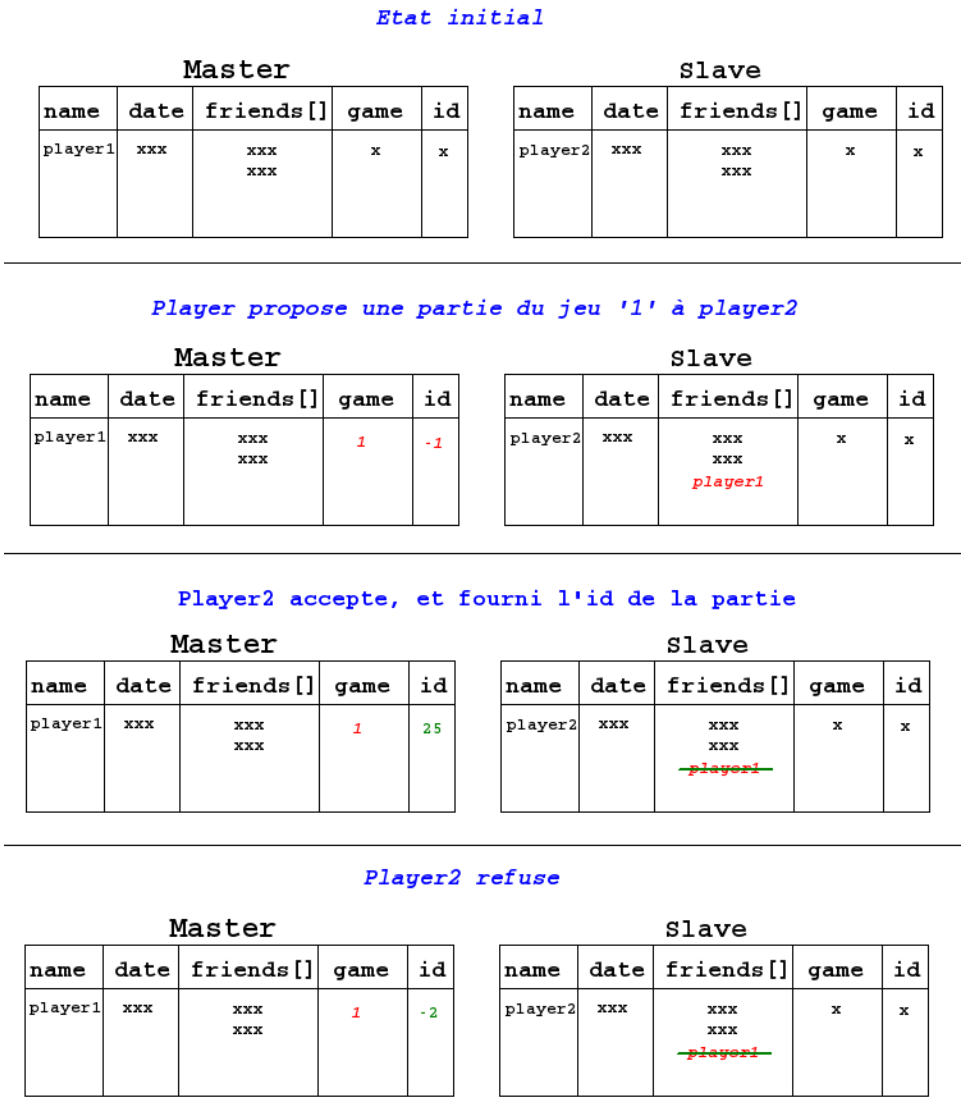

Figure 2: Amorçage d'une partie

3.4 Déroulement d'une partie de Revenge

La carte est créé via la fonction *initMap()*.
Si on est "master", token est pris au hasard.

Si on a le jeton, on récupère les coordonnées du coup joué, via la fonction *getCoo()*.
Si les coordonnées sont (-1,-1), aucun coup n'a été joué (début de partie ou impossibilité de jouer).
Sinon, on rafraichi la carte via la fonction *refresh()* qui renvoie le nombre de coups restants dans le jeu.

On vérifie qu'on peut jouer via la fonction *coupPossible()*.
Si on ne peut pas, on met les coordonnées (-1,-1) via la fonction *setCoo()*.
Si on peut, on obtient les coordonnées choisies via la fonction *choice()*, puis on place ces coordonnées via la fonction *setCoo()*.
Si on est "master, on donne le jeton à "slave", si on est "slave", on passe le jeton à "master".

Si le nombre de coups restants dans le jeu > 0, on attend le jeton pour jouer, sinon la partie est fini.

3.5 Fonctions utilisées

3.5.1 Fonctions orientées SQL

int newGame ()
Insert une nouvelle partie dans la table "revenge" et renvoie son id.
Coordonnee getCoo ()
Obtient les coordonnées x et y de la table.
void setCoo (Coordonnee coo)
Place les coordonnées x et y dans la table.
boolean getToken()
Obtient la valeur du jeton dans la table.
void setToken(boolean token)
Place le jeton 'token' dans la table.

3.5.2 Fonctions orientées Client

int initMap()**
Renvoi une map initialisée.
int refresh(Coordonnee coo)
Rafraichi la map en fonction d'un nouveau coup.
int coupPossible()
Renvoi le nombre de coups possibles.
coordonnee choice()
Invite le joueur à choisir une case.