

AbaNet : Un jeu d'abalone en réseau en Flash/PHP

Yann Ponty

25 avril 2005

1 Abalone : Règles succinctes

Le jeu d'abalone est un jeu *très simple* qui se joue sur un plateau hexagonal, où on trouve deux variétés de billes : les rouges et les noirs. Les joueurs jouent chacun leur tour, les noirs commençant après un tirage au sort usuel. A chaque tour, un joueur a le droit de déplacer 1, 2 ou 3 billes disposées en colonne de sa couleur dans n'importe quelle direction libre. Si une colonne de billes adverses inférieure est située à l'extrémité de la colonne constitué, formant ainsi un obstacle au déplacement de la colonne, qu'elle est suivie d'une case libre et de taille inférieure à celle du joueur ayant la main, alors celle ci est déplacée, une bille tombant éventuellement hors du plateau. Le gagnant est le premier qui arrive à pousser 6 des billes adverses hors du plateau de jeu.

De plus amples détails sont disponibles à l'adresse :

http://uk.abalonegames.com/rules/basic_rules/official_rules.html

2 But du projet

Le but du projet est de concevoir, réaliser puis coder un jeu d'Abalone en réseau permettant à deux joueurs humains de s'affronter. L'interface utilisateur sera codée en [Flash](#), qui permet d'obtenir assez rapidement des interfaces esthétiques. Cependant, ce langage ne permet pas la connexion directe à un autre ordinateur, via une interface de type [Socket](#), par exemple. Il permet cependant le chargement de contenu dynamique via [XML](#) ou des pages [PHP](#). Le projet s'articulera autour d'un serveur [Apache/MySQL](#) par lequel transiteront les données, selon le modèle de la figure 1.

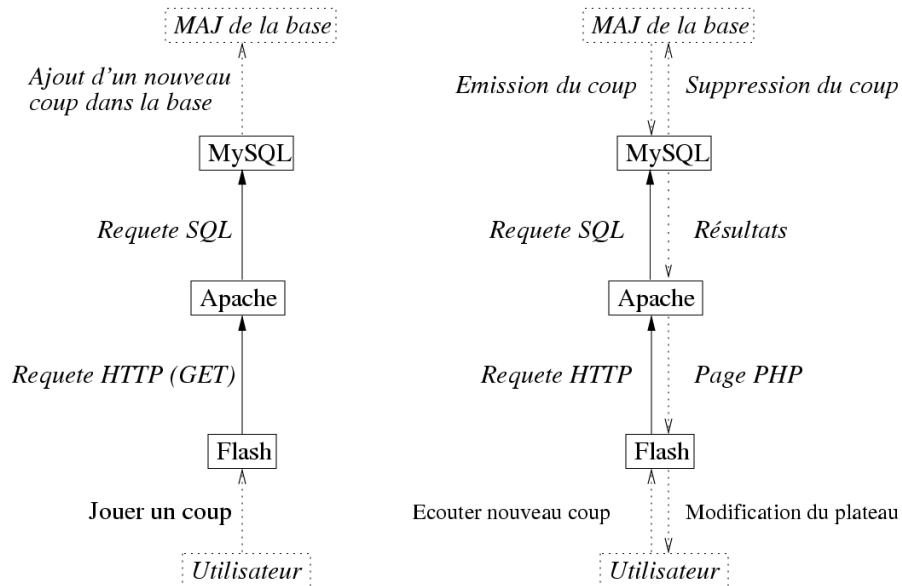


FIG. 1 – Mise à jour de la base sur jeu d'un coup valide

3 Scénarios d'utilisation

Après une phase d'initialisation préalable, il y a à tout instant un joueur *actif* et un joueur *passif*. Le joueur passif *écoute*, dans l'attente d'un coup joué par le joueur *actif*. L'interface du joueur *actif* permet à l'utilisateur de *jouer un coup valide*.

- *Ecouter* : La communication réseau est simulée par l'utilisation d'un serveur Apache, lui même connecté à une base de donnée MySQL. Après avoir bloqué toute possibilité d'interaction entre l'utilisateur et l'interface¹, l'application charge régulièrement grâce à une instance correctement configurée de la classe `LoadVars` le contenu d'une page `listen.php` générée dynamiquement. Les informations permettant d'identifier la partie concernée parmi les parties en cours seront codées et transmises dans l'url. Ex. :

`http://monsite.fr/unrep/listen.php?idpartie=23`

La requête de l'url ci dessus auprès du serveur Apache doit solliciter l'état de la partie d'identifiant 23. La réponse (i.e. la génération de la page php) devra contenir un codage de l'absence de coup joué, ou

¹Sauf éventuellement par le biais d'un bouton *Quitter*

bien un indicateur de la présence d'un coup, suivi d'une description du coup joué par l'adversaire. Cette description sera générée et codée selon un format similaire au passage des variables dans l'url. Au niveau du serveur Apache, un script **PHP** est interprété par le serveur, qui accède à la base de donnée et teste la présence d'un coup, grâce à des fonctions `mysql_connect,mysql_query ...` En fonction du résultat des requêtes décrites ci-dessus, l'interface reste en mode *passive* (pas de coup joué) ou répercute le coup joué par l'adversaire dans l'interface et passe en mode *active*, acceptant désormais les ordres de l'utilisateur(joué).

- *Jouer* : Cette fois, la communication est unidirectionnelle. L'interface, après s'être assurée de la validité du coup proposé par le joueur, prévient son adversaire par le biais du serveur Apache. Pour cela, elle sollicite auprès du serveur Apache la création d'une page `play.php` en lui fournissant dans l'url invoquée les coordonnées du coup joué. La page **PHP** contient un script qui publie le coup joué dans la base de données après avoir récupéré ses coordonnées dans le tableau `$_GET[...]`. Une fois le coup publié, l'interface rentre en mode *passif*.

4 L'encodage des variables pour **PHP** et **LoadVars**

L'**encodage URL** des variables obéit à un mécanisme très simple :

- L'url est suivie d'un *point d'interrogation* ?
- Chaque variable de nom X et de valeur Y devient dans l'url : X=Y
- Chaque déclaration de variable est séparée de la suivante par un caractère *et commercial* &

Ex. : L'encodage de variables nom, x, y, z et `estValide` de valeurs respectives joe, 3, 7, 12 et `true` donne l'url suivante :

```
http://monsite.fr/page.php?nom=joe&x=3&y=7&z=12&estValide=true
```

Remarque 1 : Il n'y a aucune notion de typage dans l'encodage URL. En particulier, il est impossible d'encoder directement des types structurés. On doit alors passer par un codage des différents champs de la variable.

Remarque 2 : Un certain nombre de caractères sont susceptibles d'interférer avec l'analyse par le serveur de l'url, ils sont donc interdits. En particulier, le &, le = et le caractère espace sont interdits et doivent être remplacés par des %XX. En outre, tous les caractères accentués n'existent pas. Des informations plus *systématique* sont disponibles à l'url :

```
http://www.blooberry.com/indexdot/html/topics/urlencoding.htm
```

5 La classe LoadVars

La classe `LoadVars` de [Flash](#) implémente un rapatriement non-bloquant des données. C'est à dire qu'elle n'attend pas que la page soit chargée ou qu'une erreur arrive pour rendre la main. Les actions à réaliser après le chargement de la page devront donc être implémentée dans une méthode surchargée de l'instance. Ex. :

```
var i = 1882658;
var c = new LoadVars();
c.onLoad = function()
{
    trace("La page a été chargée avec succès !");
    trace("Nb="+this.name+" \n");
};
c.load("userInfo.php?i="+i);
```

La fonction `c.onLoad()` est appelée dès que le contenu de la page `userInfo.php` aura pu être chargée. Si cette dernière contient une définition pour la variable `nom`, comme on peut s'y attendre, alors celui ci sera affiché par la méthode `trace` de [Flash](#).