Rémi Taunay
Vincent Brebion

Mini-projet : Mastermind

Objectif

À travers un mini-projet développé en PHP (réalisation d'un jeu, le Mastermind), seul ou par binôme, progresser en développement web côté serveur.

Contraintes

- Le jeu réalisé doit pouvoir être installé sur n'importe quel serveur web. Ainsi, aucun lien absolu ne doit être utilisé.
- Il doit comprendre un SGBD en relation avec une base de données MySQL, laquelle permet de stocker des statistiques sur les différentes parties ayant été jouées.
- Le développement doit respecter le patron MVC, lequel offre une certaine modularité à l'application.
- Une utilisation éventuelle de javascript doit porter uniquement sur l'affichage.

Plan du rapport

Mini-projet : Mastermind

Brebion - Taunay

I / Conception de l'application

1.1) Diagramme de classes

Le projet s'articule autour du modèle MVC (Modèle Vue Contrôleur). À la racine du projet il y a un dossier pour les contrôleurs, un autre pour les modèles et enfin un pour les vues. Le fichier index.php est volontairement en dehors de ces dossiers puisqu'il s'agit de la page par laquelle transitent l'intégralité des requêtes.

La partie contrôleur comprend les fichiers :

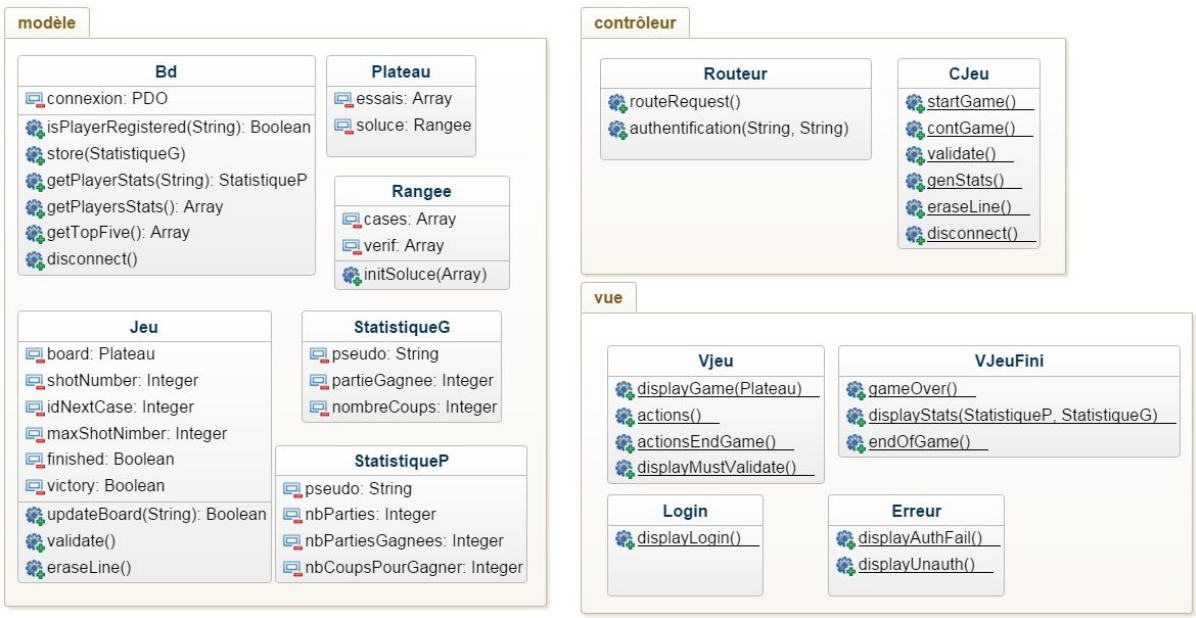
→ Routeur.php
→ CJeu.php

La partie modèle comprend les fichiers :

→ Bd.php
→ Jeu.php
→ Plateau.php
→ Rangee.php
→ StatistiquesG.php
→ StatistiquesP.php

La partie vue comprend les fichiers :

→ Erreur.php
→ Login.php
→ VJeu.php
→ VJeuFini.php
→ un sous-dossier contient les pages de style et les images utilisées



1.2) Déroulement général

L'utilisateur arrive sur la page index. Celle-ci a un rôle unique : instancier un routeur. Le routeur s'occupe alors de diriger les requêtes. Les requêtes sont émises par les vues lorsque l'utilisateur interagit avec. Il faut garder à l'esprit que lors de chaque requête, la page d'index génère un routeur, lequel va être construit. Puis, en fonction des variables de session disponibles, va reprendre le jeu au point précédent ne prenant en compte la dernière requête ayant été émise. De cette manière, le jeu progresse, en dépit de la réinstanciation du routeur à chaque requête par l'index.

Avant toute chose, l'utilisateur doit s'authentifier. Ses identifiants sont alors vérifiés via un appel à la base de données. Cet appel est bien entendu géré au niveau d'une classe dédiée : "Bd". Cette classe n'est autre qu'un PDO, lequel contient différentes méthodes à même de récupérer des informations de la base de données ainsi que d'en stocker (historique des différentes parties ayant été jouées).

En premier lieu le routeur se charge donc de diriger le nouveau joueur vers la vue Login.php. Deux champs lui permettent alors d'entrer son pseudo et son mot de passe. Ces informations sont envoyées au routeur lorsqu'il confirme ses identifiants. La classe Bd vérifie alors si le pseudo et le mot de passe forment un couple correct dans la base de données. Le mot de passe est bien entendu crypté, on ne souhaite pas que ce dernier soit stocké en clair dans la base de données.

Dans le cas où l'utilisateur entre des informations invalides il est redirigé vers une page d'erreur. Une fois l'authentification réussie, le routeur commence une nouvelle partie et affiche le plateau initial. Le joueur peut alors visualiser le plateau de jeu interagir avec la barre de choix de couleur, en bas de l'écran.

Pendant le jeu, le routeur fait des appels au contrôleur de jeu (CJeu), en fonction des requêtes. Lors d'un appui sur "valider", la rangée en cours est comparée à la solution et les cases de vérifications sont mises-à-jour. Si la demande était d'effacer la rangée les cases de celle-ci se réinitialisent et le joueur peut de nouveau saisir les couleurs qu'il souhaite.

Lorsque la partie est finie l'utilisateur accède à une vue résumant la partie. Celle-ci comprend la solution qui était à trouver, les coups joués mais aussi des statistiques sur ce joueur (nombre de parties, taux de victoires, etc) ainsi qu'un classement des cinq meilleurs joueurs. Sur cette vue l'utilisateur peut décider de rejouer une partie ou bien se déconnecter, auquel cas il sera redirigé vers la page d'authentification.

1.3) Déroulement d'une partie

L'utilisateur a le droit de jouer dix coups pour trouver la solution du plateau.
À la validation d'un coup des cases indiqueront la validité du coup (couleurs bien placées, bonnes couleurs mais mal placées, couleurs absentes). Au-delà des dix coups disponibles, le joueur perd la partie.
Chaque coup se déroule suivant trois étapes:

Première étape : La sélection de la couleur d'une case par l'utilisateur se fait par le tableau coloré mis à disposition en dessous du plateau de jeu. Le plateau se met à jour automatiquement lors du clic sur l'une des couleurs : les cases se changent avec la couleur choisie.

Deuxième étape : Lorsqu'une rangée du plateau est remplie (quatres cases sont colorées) l'utilisateur peut la valider ou bien l'effacer s'il souhaite choisir d'autres couleurs. Si l'utilisateur essaie de sélectionner une couleur alors que la rangée est pleine un message d'erreur s'affiche lui indiquant qu'il doit valider ou effacer la rangée actuelle. En revanche si la rangée n'est pas complète l'utilisateur ne peut que l'effacer.

Troisième étape : le tableau du jeu s'affiche avec les modifications apportées par la validation des cases de la rangée. L'utilisateur prend alors connaissance de sa progression et l'on retourne à l'étape n°1.

II / Implémentation

La plupart des méthodes ne demandant pas de traitement spécifique, nous porterons l'intérêt sur un ensemble restreint de méthodes clés, dont les prototypes sont listés ci-dessous :

```
/**
* Classe : Routeur
* Méthode routant toutes les requêtes entrantes de la page index.php
*/
public function routeRequest()
```

Comme son prototype le suggère, cette méthode est celle qui fait les appels de plus haut niveau dans l'application. En effet, elle appelle les méthodes appropriées en fonction de l'avancement du jeu. Cette méthode se fie aux variables contenues dans $_SESSION et $_POST en testant leur existence à l'aide de isset().

```
/**
* Classe : Rangee
* Méthode qui initialise les cases de la rangée solution en tirant
* une couleur au hasard parmi celles du jeu
* @param $colors array Les couleurs possibles
*/
public function initSoluce($colors)
```

Cette méthode vient se substituer en partie au second joueur supposé du jeu. En effet, un de ses rôles est de choisir la combinaison mystère, en mixant les différentes couleurs à disposition. C'est ce qui est réalisé ici.

```
/**
* Classe : Jeu
* Méthode qui met à jour la correspondance essai du joueur / soluce (à
condition que le coup joué soit valide)
* Un coup est valide si quatre couleurs sont données (aucune case non
colorée)
*/
public function validate()
```

Cette méthode a incontestablement été la plus difficile à développer. En effet, s'il est aisé pour un joueur humain de déterminer le nombre de pions bien placés, le nombre de pions de bonne couleur mais incorrectement placés, ainsi que le nombre de pions faux, nous n'avons pas réussi du premier coup à implémenter un algorithme capable de reproduire ce comportement, c'est-à-dire effectuant cette validation. C'est pourquoi nous détaillons ici son fonctionnement intrinsèque :

- Il faut vérifier en premier lieu que la rangée soumise à vérification est complète, sans quoi effectuer une validation n'a aucun sens.

- Trois éléments sont ensuite nécessaires : la rangée à valider, la rangée réponse (la solution secrète), et la rangée contenant les marqueurs de vérification, qu'il faut remplir en conséquence des deux autres.
- Après les avoir récupérés, les vérifications commencent, nous les avons découpés en trois étapes successives :

    1. Les couleurs bien placées sont directement validées, car ce sont les plus simples à traiter, on les retire alors de l'ensemble solution (qui est en réalité une copie de la solution afin de ne pas l'altérer).

    2. Pour chaque couleur non traitée et présente dans l'ensemble solution, on récupère son indice dans l'ensemble solution afin de rendre cette couleur indisponible postérieurement. En effet, si plusieurs couleurs identiques sont jouées alors qu'elle n'apparaît qu'une unique fois dans la solution, le marqueur approprié ne doit être positionné que pour l'une d'entre elles.

    3. Les cases non traitées sont alors forcément des couleurs n'ayant pas leur place dans la combinaison secrète, on place les marqueurs correspondant dans la rangée de vérification.

- Pour finir, on trie la rangée de vérification afin de ne pas révéler la correspondance entre les marqueurs de celle-ci et les cases de la rangée soumise.
- On met également le statut de la partie à jour : Le jeu est-il terminé ; gagné ?

```
/**
* Classe : Bd
* Renvoie les statistiques des 5 meilleurs joueurs
* Sélectionne en fonction de deux critères, de même importance :
* le taux de victoires et le nombre de coups nécessaires pour y parvenir
* @return array Les statistiques relatives aux 5 meilleurs joueurs
*/
public function getTopFive()
```

Cette méthode ne suit pas les spécifications. En effet, elle ne se contente pas de donner les statistiques des 5 meilleurs joueurs en fonction de leur unique meilleure partie (celle gagnée en un nombre minimal de coups). L'explication réside dans le fait que son intérêt serait, à très long terme, nul. En effet, la probabilité de gagner dès le premier coup (pure chance) est de 1 (la bonne combinaison), sur 65536 (4 pions exposant 8 couleurs disponibles). Même si c'est infime, il peut arriver au bout d'un certain temps que les "meilleurs joueurs" seront les plus chanceux, y compris pour des victoires en deux ou trois coups.

Après réflexion, deux critères doivent être pris en compte pour déterminer le niveau d'un joueur : son taux de victoire et le nombre moyens de coups nécessaires pour parvenir à une victoire. Avec ces deux indicateurs, il suffit de créer un indicateur combiné. Ce procédé simule en quelque sorte un classement par Elo.

III / Prise de recul sur le déroulement du mini-projet

Cette partie fait état des enseignements que l'on a pu tirer de ce mini-projet, en dehors de l'aspect purement technique. Elle retrace chronologiquement le déroulement du mini-projet en donnant les difficultés rencontrées, et comment les prévenir lors de futurs projets.

Après avoir terminé la conception détaillée, nous nous sommes lancés dans l'implémentation. Malheureusement, beaucoup de méthodes de la partie vue avaient un type de retour "void", car ce retour s'effectue indirectement, via un formulaire. Il était donc difficile de se représenter le flux d'exécution, d'autant plus qu'ici le constructeur de Routeur est appelé à chaque requête. Le typage dynamique et faible du PHP ne facilite pas non plus la compréhension (bien que compensé en partie par l'utilisation de la PhpDoc).

C'est donc lors de cette première phase d'implémentation que les problèmes sont survenus. En effet, nous ne testions pas assez, n'ayant pas encore de base de donnée afin de vérifier la partie concernant l'authentification, par exemple. De même, les chemins du serveur web de PhpStorm fonctionnaient de manière inattendue.

Nous nous sommes alors tournés vers un serveur apache2 externe, muni d'une base de données MySQL en parallèle, ce qui a permis enfin de tester convenablement. Afin de ne pas perdre de temps dans le débogage, nous avons pris la décision de recréer le projet, en repartant "de zéro".

Dans ce dernier, nous avons opté pour l'intégration continue, c'est-à-dire que l'on incorporait, testait, et modifiait si nécessaire, au fur et à mesure, les classes et méthodes précédemment développées. Ces modifications apportaient des changements dans le diagramme de classe qui a, de ce fait, évolué tout au long de l'implémentation. Cela a permi d'avoir une application toujours fonctionnelle, dont l'avancement progressif a mené jusqu'à la version finale.