



Rapport mastermind
Programmation web coté serveur

Paul Orhon

DUT INFO 2 – G2

4 décembre 2016

Table des matières

1	Introduction	2
2	Le jeu	2
2.1	Règle de base	2
2.2	Comment jouer ?	2
3	Le code	6
3.1	Controleur	6
3.1.1	RouteurControleur	6
3.1.2	ControleurAuthentification	7
3.1.3	ControleurJeu	7
3.2	Modele	7
3.2.1	bd	7
3.2.2	jeu	8
3.2.3	Combinaison	8
3.3	Vue	8
3.3.1	VueConection	8
3.3.2	VueJeu	9
3.3.3	VueStat	9
3.3.4	VueFin	9
3.3.5	VueErreur	9
3.3.6	CSS	9
3.3.7	JS	9

1 Introduction

Pour le module de programmation web coté serveur nous avons reçue un projet, qui est de réaliser un mastermind en PHP en respectant l'architecture MVC. Nous avons eu le droit d'utiliser du CSS et JS pour le style des pages.

Tout le contenu est disponible sur GitHub : <https://github.com/paul604/Mastermind>.

2 Le jeu

On nous a donc demandé de créer un mastermind.

2.1 Règle de base

wikipedia (<https://fr.wikipedia.org/wiki/Mastermind#Principe>) :

« Un joueur commence par placer son choix de pions sans qu'ils soient vus de l'autre joueur à l'arrière d'un cache qui les masquera à la vue de celui-ci jusqu'à la fin de la manche.

Le joueur qui n'a pas sélectionné les pions doit trouver quels sont les quatre pions, c'est-à-dire leurs couleurs et positions.

Pour cela, à chaque tour, le joueur doit se servir de pions pour remplir une rangée selon l'idée qu'il se fait des pions dissimulés.

Une fois les pions placés, l'autre joueur indique :

1. le nombre de pions de la bonne couleur bien placés en utilisant le même nombre de pions noir ;
2. le nombre de pions de la bonne couleur, mais mal placés, avec les pions blancs.

Il arrive donc surtout en début de partie qu'il ne fasse rien concrètement et qu'il n'ait à dire qu'aucun pion ne correspond, en couleur ou en couleur et position.

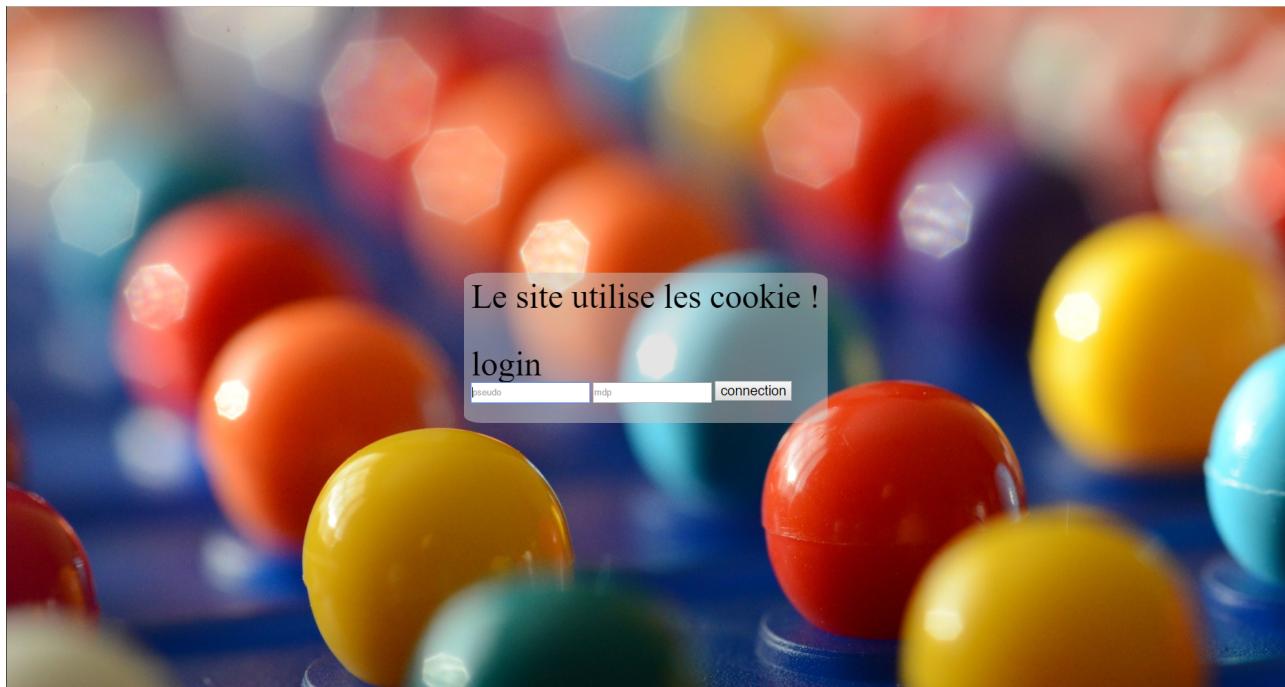
La tactique du joueur actif consiste à sélectionner en fonction des coups précédents, couleurs et positions, de manière à obtenir le maximum d'informations de la réponse du partenaire puisque le nombre de propositions est limité par le nombre de rangées de trous du jeu. Dans la plupart des cas, il s'efforce de se rapprocher le plus possible de la solution, compte tenu des réponses précédentes, mais il peut aussi former une combinaison dans le seul but de vérifier une partie des conclusions des coups précédents et de faire en conséquence la proposition la plus propice à la déduction d'une nouvelle information.

Le joueur gagne cette manche s'il donne la bonne combinaison de pions sur la dernière rangée ou avant. Dans tous les cas, c'est à son tour de choisir les pions à découvrir. Mais il est interdit de mettre une couleur en double, en triple ou en quadruple aussi bien dans les pions secrets que dans les pions "publics". »

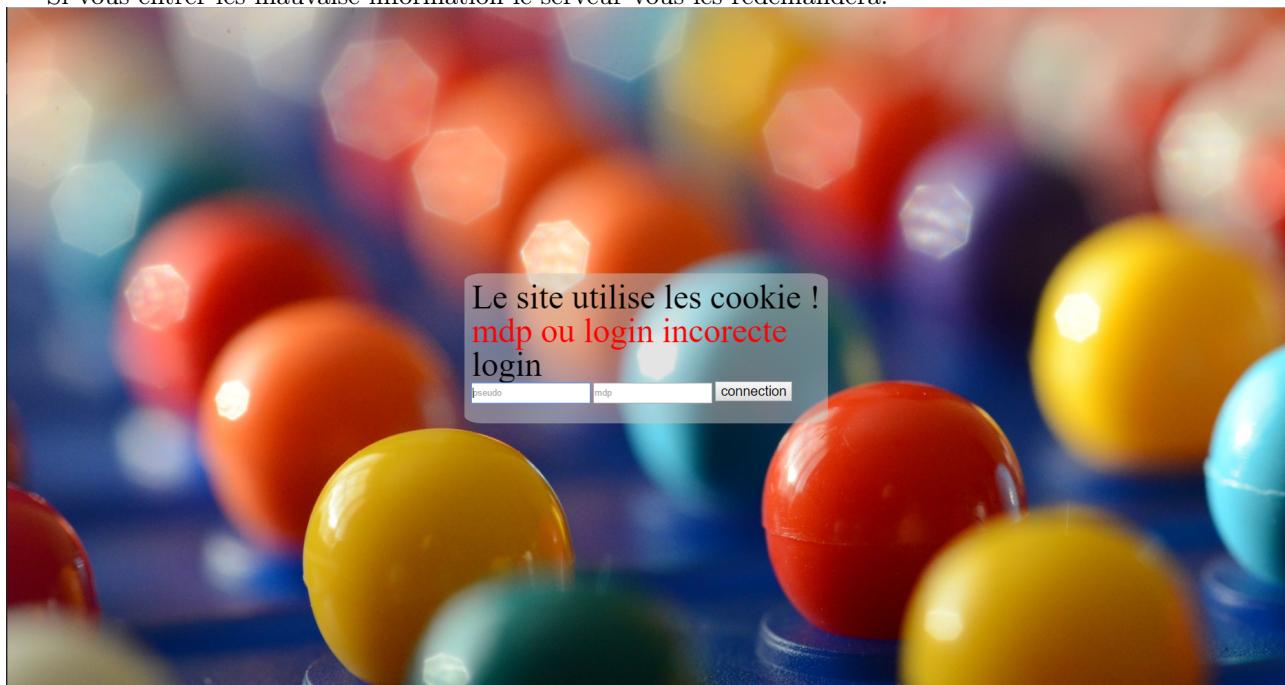
Ici, le joueur qui place les pions cacher est remplacé par l'ordinateur.

2.2 Comment jouer ?

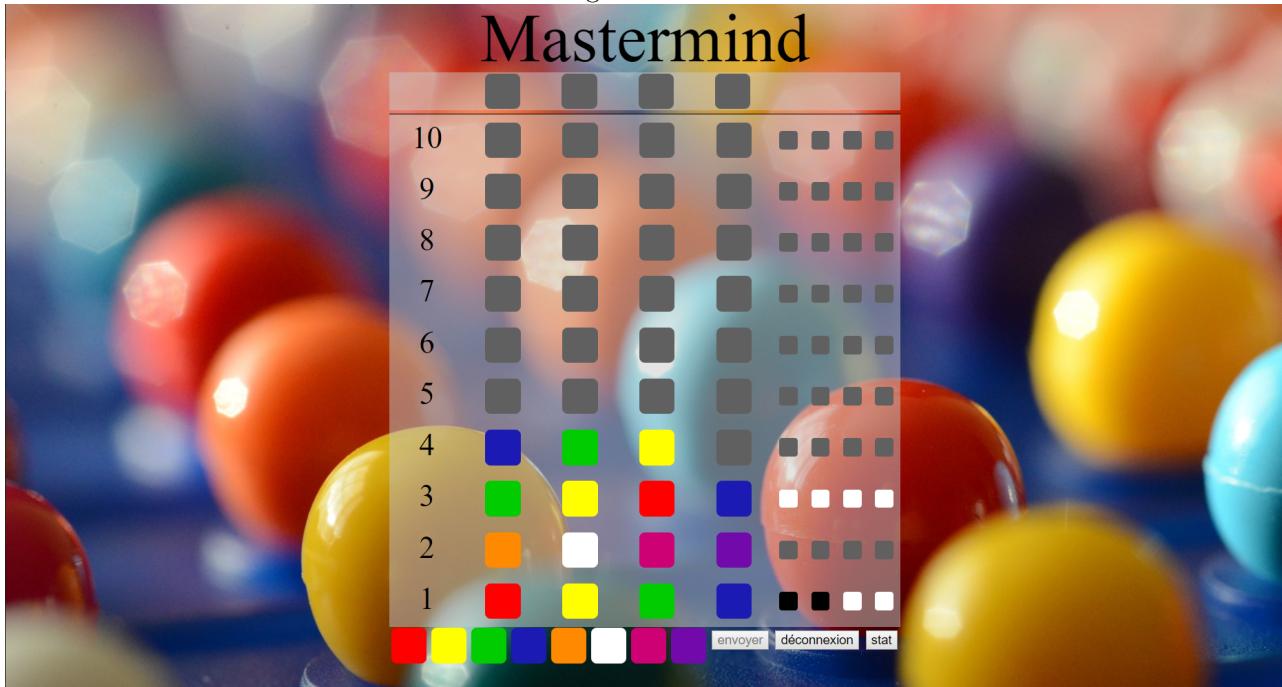
Dans un premier temp il va falloir se connecter. (il y a seulement deux compte : titi(mdःp :titi) et toto(mdःp :toto))(Les images sont susceptible d'avoir changé)



Si vous entrer les mauvaise information le serveur vous les redemandera.



Une fois connecté vous tomberez sur cette image :



Pour jouer il vous suffit de cliquer sur l'une des couleurs présente dans la ligne de choix (à gauche du bouton envoyer, elle se placera à la première place trouver), ou sur l'une des couleurs déjà placée sur le plateau (elle se placera à la même place où elle est placé mais sur la ligne courante).

Une fois la bonne combinaison trouvée ou le nombre maximum de coup atteint la solution se dévoile et vous pouvez faire une partie.



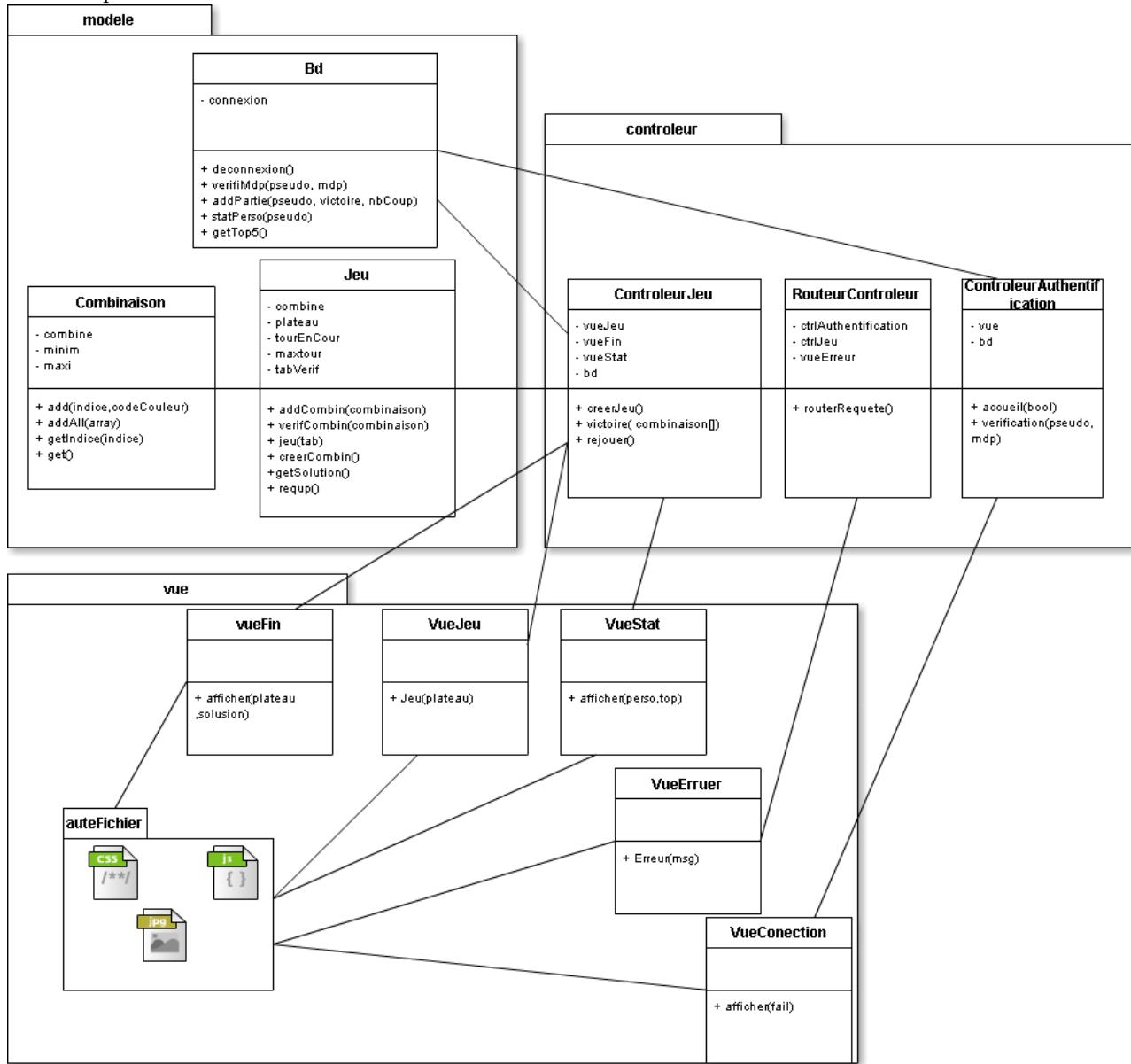
Quand vous vous êtes connecter vous pouvez voir les statistique personnel et les 5 meilleur partie.



3 Le code

Ici je vais expliquer le code et les choix réalisé.

Pour le respect de l'architecture MVC on peu voir quelle est respecter en regardant le diagramme de class qui nous montre que les contrôleurs font la passerelle entre le modèle et la vue et qu'il y a aucun échange entre ces deux partie.



3.1 Controleur

3.1.1 RouteurControleur

RouteurControleur est la class qui décide qui appeler pour effectuer les action.

Tout le code est entouré dans un *try catch*, se qui permet de détecter tout problème, de détruire la session, et de diriger le joueur vers la page d'erreur si une exception est levé n'importe où dans le code.

Dans la fonction *routerRequete()*, il y a une suite de condition qui sont ordonné dans un sens précis pour éviter tout bug.

Pour éviter le problème avec le recharge de la page (F5) des redirection ("header("Location : index.php");") sont présent dans certaine condition.

Premier if : Le joueur demande de se connecter, puis après vérification dans la base de données et affectation du cookie de session le joueur est redirigé.

Deuxième if : Le joueur demande de se déconnecter, on supprime la session et on le renvoie à la page de connections.

Troisième if : Le joueur a fait une proposition de combinaison, on rentre sa proposition dans une session puis on le redirige pour éviter le problème du F5.

Quatrième if : Le joueur demande de rejouer, on remet le jeu à zéro, on en crée un nouveau et on le redirige comme à sa connections.

Cinquième if : Le joueur demande de voir les stat.

Sixième if : Le joueur a fini la partie, on affiche la vue de fin.

Septième if : Le joueur passe par ici à la suite de la redirection du troisième if.

Huitième if : Le joueur arrive ici quand il commence une nouvelle partie et quand il a quitté la partie et qu'il revient.

else : On affiche la page de connections.

3.1.2 ControleurAuthentification

ControleurAuthentification s'occupe de l'identification du joueur.

La fonction *accueil(\$bool)* affiche la page de connection. Si *\$bool* est à *true*, le joueur a déjà tenté de se connecter (permet l'affichage d'un message erreur).

La fonction *verifCo(\$pseudo, \$mdp)* permet la vérification du couple pseudo, mdp en appelle la base de données. Si tout est bon on affecte le pseudo à un cookie de session et on retourne *true* sinon on retourne *false*.

3.1.3 ControleurJeu

ControleurJeu s'occupe du jeu.

__construct() est un peu particulier car si il n'y a pas encore de *jeu* dans le cookie de session on en créer un nouveau. Cela permet de toujours avoir le même *jeu* pour un joueur et permet de ne pas remettre à zéro le plateau apprêt chaque coup.

toStringPlateau(\$plateau) permet d'envoyer le plateau à la vue sans problème, c'est l'équivalent du *toString* en java.

jeu(\$tab) est la fonction principale de la classe. En fonction de l'entrée elle décide quoi faire. Si *\$tab* est null on créer un nouveau jeu, si on veux reprendre la partie ou si on vient de jouer un coup.

victoire() permet de récupérer le plateau et la solution pour l'afficher au joueur.

stat() permet d'afficher les stat en consultant la base de données.

3.2 Modele

3.2.1 bd

bd est la classe qui s'occupe de la communication avec la base de données.

verifiMdp(\$pseudo, \$mdp) est la fonction qui permet de vérifier le couple pseudo, mdp. Si *\$pseudo* est présent dans la base de données on vérifie si *\$mdp* correspond au mot de passe trouvé (apprêt cryptage de *\$mdp*).

addPartie(\$pseudo, \$victoire, \$nbCoup) permet d'ajouter la partie finie à la base de données. *\$victoire=0* si la partie a été perdue et 1 si la partie a été gagnée.

statPerso(\$pseudo) cette fonction permet de récupérer les statistiques personnelles d'un joueur notamment ses parties gagnées et le nombre total de parties jouées.

getTop5() récupère les 5 meilleur parti de tout les joueur confondu et retourne leur pseudo et leur nombres de coups.

3.2.2 jeu

jeu est la class de basse du jeu. Avant tout on vérifie si il y a un cookie se session, si se n'ait pas le cas on démarre la session. Les couleur son symboliser par des numéro pour faciliter les vérification

addCombin(\$combinaison) permet l'ajout d'une combinaison dans le plateau.

verifCombin(\$combinaison) permet de vérifier la proposition du joueur.

Dans la première boucle on vérifie si les couleur de la proposition sont bien placer. Si c'est le cas on ajoute le numéro de leur, place dans la proposition, dans un tableau "fait" pour ne pas les retester par la suite. Si tout les couleur sont bien placer on retourne *true*.

Dans la deuxième boucle on vérifie si les couleur de la proposition sont mal placée, mais sans revérifier les couleur déjà présente dans le tableau "fait". Si une couleur est mal placée on ajoute sont indice dans un tableau "fait2" pour ne pas la retester si la couleur est présente plusieurs fois dans la proposition.

jeu(\$tab) est la fonction principale. Si \$tab est un tableau alors le joueur a fait une proposition donc on fait les vérification, sinon c'est la création du jeu et si le *tourEnCour* est différent de 0 il y a un problème. Ensuite on incrémenter *tourEnCour* de 1, puis si il a pas dépasser le nombre de coups maximum on retourne *false* pour indiquer la fin de la partie, sinon retourne tous se que la vue aura besoin : le plateau, la table de vérification (les pion bien et mal placé), et *tourEnCour*.

creerCombin() permet la création de la combinaison à trouver en utilisant un random.

getSolution() est le getter de *\$this->combine* qui est la combinaison à trouver.

requp() est appeler si le joueur vue récupère la parti en cour en retournant tous se que la vue aura besoin : le plateau, la table de vérification (les pion bien et mal placé), et *tourEnCour*.

3.2.3 Combinaison

Combinaison est la class qui symbolise une proposition/combinaison.

__construct(\$verif) si \$verif est à 0 alors *Combinaison* sait que les valeur à accepter sont entre 1 et 8 et corresponde à une proposition/combinaison dans le *plateau*, sinon si \$verif est à 1 alors *Combinaison* sait que les valeur à accepter sont entre -2 et 0 et corresponde à une vérification proposition (0=> la couleur n'existe pas ; -1=> la couleur existe mais est mal placée ; -2=> la couleur est bien placée).

add(\$indice, \$codeCouleur) permet d'ajouter une couleur à un indice précis. Mais avant d'ajouter la couleur il vérifie elle est comprise dans le bonne intervalle.

addAll(\$array) correspond au setter de *\$this->combine*. Elle permet d'ajouter un tableau, de taille 4, contenant des couleur.

getIndice(\$indice) permet de récupérer une couleur présent à un indice précis.

getall() et *get()* corresponde au getter de *\$this->combine* (les deux sont présent car *getall()* permet de vérifier le fonctionnement de *getIndice(\$indice)*)

3.3 Vue

3.3.1 VueConection

VueConection gère la vue de connections via la fonction *afficher(\$fail)*, si \$fail est à *true* alors le joueur à déjà tenter de se connecter et on lui affiche un message. Pour les champ de connections un formulaire en post.

3.3.2 VueJeu

VueJeu gère la vue de jeu via la fonction *Jeu(\$plateau)*. *\$plateau* est un tableau constituer de deux tableaux et d'une valeur. La valeur correspond au tour courant, le premier tableau est le plateau de jeux (là où le joueur fait ses proposition) et le deuxième tableau correspond au tableau de vérification (là où est afficher si les pions de couleur sont bien/mal placée).

A la fin ce trouve un formulaire (utilisant la méthode post) qui permet de faire sont choix. Ce formulaire est cacher au joueur et est modifier avec du Java Script. Il y a aussi un bouton pour ce déconnecter et un pour voir les statistiques.

3.3.3 VueStat

VueStat permet d'afficher les statistiques personnel du joueur et le top 5 des partie de tout les joueur avec la fonction *afficher(\$perso,\$top)*. *\$perso* est un tableau qui contient le nombre de partie gagné et le nombre de partie jouer. *\$top* est un tableau de 5 élément qui contienne un pseudo et le nombre de coups utiliser pour gagné la partie. Un bouton permet de retourner joueur et un autre de se déconnecter.

3.3.4 VueFin

VueFin est appeler, en utilisant *afficher(\$plateau,\$solusion)*, quand le joueur a fini la partie. *\$plateau* est constituer des même objet que dans *VueJeu* (cf.3.3.2). *\$solusion* est un tableau de 4 chiffre représentant les couleur a trouver.

A la fin ce trouve des boutons permettant de rejouer, ce déconnecter et pour voir les statistiques. Les boutons sont dans des formulaire différent pour faciliter la mise en page.

3.3.5 VueErreur

VueErreur est la vue qui gère les erreur. Elle est appeler quand on rentre dans l'un des *catch* de *RouteurControleur* (cf.3.1.1) en utilisant la fonction *Erreur(\$msg)*. *\$msg* correspond au message de l'erreur.

3.3.6 CSS

Il y a deux css, un qui effectue une remise à zéro des règle et un autre qui effectue la mise en forme.

3.3.7 JS

Le JS est utiliser pour remplir le formulaire invisible (cf.3.3.2)

set(color) permet d'ajouter la couleur choisie dans le tableau et dans le formulaire.

unSet(i) permet de retirée la couleur choisie dans le tableau et dans le formulaire.

up(indice,color) permet d'ajouté une couleur placée précédemment dans le tableau et dans le formulaire.

submit(b) permet de verrouille/déverrouille le bouton envoyer.