TP PHP - Mini jeu de combat

Intro

Ce que nous allons réaliser est très simple. Nous allons créer une sorte de jeu. Chaque visiteur pourra créer un personnage (pas de mot de passe requis pour faire simple) avec lequel il pourra frapper d'autres personnages.

Le personnage frappé se verra infliger un certain degré de dégâts. Un personnage est défini selon 2 caractéristiques :

- Son nom (unique).
- Ses dégâts.

Les dégâts d'un personnage sont compris entre 0 et 100. Au début, il a bien entendu 0 de dégât. Chaque coup qui lui sera porté lui fera prendre entre 3 et 10 points de dégâts. Une fois arrivé à 100 points de dégâts, le personnage est mort (on le supprimera alors de la BDD).

Pré conception

Avant de nous attaquer au cœur du script, nous allons réfléchir à son organisation. De quoi aura-t-on besoin ? Puisque nous travaillerons avec des personnages, nous aurons besoin de les stocker pour qu'ils puissent durer dans le temps. L'utilisation d'une base de données sera donc indispensable.

Le script étant simple, nous n'aurons qu'une table personnages qui aura différents champs. Pour les définir, réfléchissez à ce qui caractérise un personnage. Ainsi nous connaissons déjà 2 champs de cette table que nous avons définis au début : nom et dégâts . Et bien sûr, n'oublions pas le plus important : l'identifiant du personnage ! Chaque personnage doit posséder un identifiant unique qui permet ainsi de le rechercher plus rapidement (au niveau performances) qu'avec son nom.

Vous pouvez donc ainsi créer votre table tous seuls via PhpMyAdmin. Si vous n'êtes pas sûrs de vous, je vous laisse le code SQL créant cette table :

```
DROP TABLE IF EXISTS `personnages`;
CREATE TABLE IF NOT EXISTS `personnages` (
  `id` smallint UNSIGNED NOT NULL AUTO_INCREMENT,
  `nom` varchar(50) CHARACTER SET utf8mb4 COLLATE utf8mb4_general_ci NOT
NULL,
  `degats` int UNSIGNED NOT NULL DEFAULT '0',
  PRIMARY KEY (`id`),
```

```
  UNIQUE KEY `nom` (`nom`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1 COLLATE=latin1_general_ci;
```

Partie 01 : Le personnage

Nous allons commencer le TP. Pour rappel, nous allons réaliser un petit jeu mettant en scène des personnages qui peuvent combattre. Qui dit personnages dit objets Personnage (je pense que ça, vous l'avez deviné, puisque nous avons travaillé dessus durant les premiers chapitres).

Arrive maintenant un moment délicat dans votre tête : « Par où je commence ? ». Pour construire une classe, vous devez répondre à deux questions qui vont vous permettre d'établir le plan de votre classe :

- Quelles seront les caractéristiques de mes objets ?
- Quelles seront les fonctionnalités de mes objets ?

Voici comment procéder. Nous allons dans un premier temps dresser la liste des caractéristiques du personnage pour ensuite se pencher sur ses fonctionnalités

```
Rappel : le traitement des données (c'est-à-dire l'exécution de requêtes
permettant d'aller effectuer des opérations en BDD) se fera dans une autre
classe. Ne vous en préoccupez donc pas maintenant !
```

Nous avons déjà défini une partie des infos de notre Personnage avec notre table. Nous allons pouvoir faire un miroir avec les attributs dans notre classe.

Créez donc dans un fichier `Personnage.php` la classe Personnage avec ses différents attributs. Dans un soucis d'encapsulation, les attributs seront privés, puis auront des getters et des setters.

Ensuite, déterminons les fonctions que peut faire notre classe.

Un personnage doit pouvoir :

- Frapper un autre personnage
- Recevoir des dégâts.

À chaque fonctionnalité correspond une méthode. Écrivez ces méthodes dans la classe en mettant en commentaire ce qu'elles doivent faire. Vous coderez les fonctions après.

Petites questions à vous poser à vous mêmes :

- Puis-je me frapper moi même ?
- Que doivent renvoyer mes méthodes, si elle doivent renvoyer quelques chose ?
- Quels seront les paramètres de mes méthodes ?

Pour la première, la réponse est simple, il suffira de ne rien faire si le personnage est lui même.

Pour la seconde, je vous propose de définir des constante qui seront des code chiffré (1,2,...). Voici une proposition :

- Méthode frapper() : la méthode renverra la valeur de la constante CEST_MOI ;
- Méthode recevoirDegats() : la méthode renverra la valeur de la constante PERSONNAGE_TUE si les dégâts dépasse une valeur maximale (100)
- Méthode recevoirDegats() : la méthode renverra la valeur de la constante PERSONNAGE_FRAPPE si l'ennemie n'est pas tué.

Votre classe devrait contenir ses constante ou bien alors créer une enum (si votre version de php > 8.1)

```
const CEST_MOI = 1 ;
const PERSONNAGE_TUE = 2 ;
const PERSONNAGE_FRAPPE = 3 ;
```

Et pour finir, les paramètres. Pour frapper(), il semble requis d'avoir un objet Personnage en tant que cible. Puis, pour recevoirDegats(), vu que nous infligeons des dégats aléatoires, nous pouvons laisser la méthode sans paramètres et les calculer dans la fonction.

Pour finir et éviter les nombres magiques, je vous invite a créer des variables qui contiendront les informations numériques de notre Personnage => PV_MAX, DEGATS_MIN, DEGATS_MAX.

Il ne vous reste plus qu'à coder les fonctions ! Bon courage.

```
Info : Si vous connaissez le principe d'hydratation, je vous invite à créer
et utiliser une fonction hydrate($data) qui vous sera utile.
```

Partie 02 : Stockage en base de donnée

Attaquons nous maintenant à la deuxième grosse partie de ce TP, celle consistant à pouvoir stocker nos personnages dans une base de données. Grande question maintenant : comment faire ?

Au cas où certains seraient toujours tentés de placer les requêtes qui iront chercher les personnages en BDD dans la classe Personnage, je vous arrête tout de suite et vous fais un bref rappel avec cette phrase que vous avez déjà rencontrée : une classe, un rôle.

La classe Personnage a pour rôle de représenter un personnage présent en BDD. Elle n'a en aucun cas pour rôle de les gérer. Cette gestion sera le rôle d'une autre classe,

communément appelée manager (ou bien DAO pour Data Access Object). Dans notre cas, notre gestionnaire de personnage sera tout simplement nommée `PersonnageManager`.

Structure

Notre classe ne possèdera qu'un attribut privé `$db`. Celui n'ayant pas de raison de changer, il sera privé et ne sera mis à jour que dans le constructeur.

Fonctionnalités

Notre manager va pouvoir faire toute une liste de chose :

- Enregistrer un nouveau personnage
- Modifier un personnage
- Supprimer un personnage
- Sélectionner un personnage
- Compter le nombre de personnages
- Récupérer une liste de plusieurs personnages
- Savoir si un personnage existe.

Cela nous fait ainsi 7 méthodes à implémenter !

```
Comme d'habitude, écrivez le nom des méthodes en ajoutant des commentaires
sur ce que doit faire la méthode. Cela permet de mieux concevoir votre code.
```

Normalement, l'écriture des méthodes devrait être plus facile que dans la précédente partie. En effet, ici, il n'y a que des requêtes à écrire : si vous savez utiliser PDO, vous ne devriez pas avoir de mal !

Pour vous donner un coup de pouce, voici comment faire une requête préparé

```php
<?php
$stmt = $dbh->prepare("SELECT * FROM REGISTRY where name = ?");
// Bind les ? dans la requete avec une liste de params
$stmt->execute([$_GET['name']]);
foreach ($stmt as $row) {
        print_r($row);
}
?>
```

Partie 03 : Mise en place du visuel

J'ai le plaisir de vous annoncer que vous avez fait le plus gros du travail ! Maintenant, nous allons juste utiliser nos classes en les instanciant et en invoquant les méthodes souhaitées sur nos objets. Le plus difficile ici est de se mettre d'accord sur le déroulement du jeu.

Celui-ci étant simple, nous n'aurons besoin que de 2 fichiers. Commençons par le début : que doit afficher notre mini jeu lorsqu'on ouvre la page pour la première fois ? Il doit afficher un petit formulaire nous demandant le nom du personnage qu'on veut créer ou utiliser.

Comme cela sera notre page d'accueil, je vous invite à créer un fichier `index.php`. L'extension .php permet quand même d'utiliser de l'HTML directement.

Version sans CSS du formulaire

Réalisez ce formulaire. Il utilisera la méthode POST et l'action visera un fichier nommé `gestion-perso.php`

Vient ensuite la partie traitement. Deux cas peuvent se présenter :

- Le joueur a cliqué sur Créer ce personnage. Le script devra créer un objet Personnage en passant au constructeur un tableau contenant une entrée (le nom du personnage). Il faudra ensuite s'assurer que le personnage ait un nom valide et qu'il n'existe pas déjà. Après ces vérifications, l'enregistrement en BDD pourra se faire.
- Le joueur a cliqué sur Utiliser ce personnage. Le script devra vérifier si le personnage existe bien en BDD. Si c'est le cas, on le récupère de la BDD

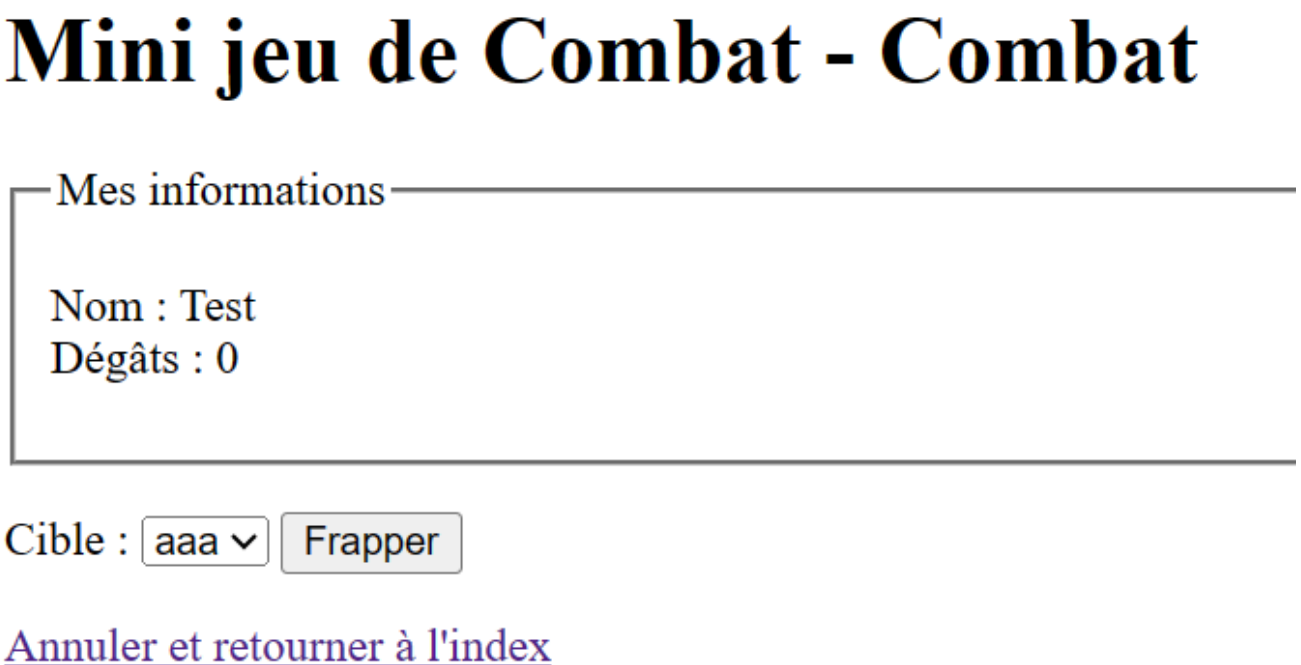
```
Un nom valide sera juste une nom n'étant pas vide. Vous pouvez adapter ce
comportement si besoin.
```

Cependant, avant de faire cela, il va falloir préparer le terrain. Dans `gestion-perso.php` il faudra :

- Charger nos 2 classes.
- Une instance de PDO devra être créée.
- Une instance de notre manager devra être créée.

A vous de jouer pour vérifier que les données envoyé par index.php à gestion-perso.php sont valides et permettent de récupérer/créer un personnage. Si une erreur survient, je vous invite à afficher un message d'erreur puis un lien renvoyant sur l'index.

Si tout est bon, vous avez votre personnage. Il est temps de construire un deuxième formulaire. Celui-ci aura besoin de données au préalable. Il vous faudra donc récupérer la liste des Personnages différents du notre (Pour éviter de se taper soi-même, c'est plus sympa). Notre formulaire sera aussi avec la méthode POST mais l'action sera le fichier lui même.



Version sans CSS du 2ème formulaire

A vous de coder votre formulaire. il faudra aussi affiché les information du personnage directement sur la page. Cela permet de voir les dégâts. Le champs cible contient la liste des cibles potentielles.

Il reste maintenant une dernière partie à développer : celle qui s'occupera de frapper un personnage. Puisque nous avons déjà écrit tout le code faisant l'interaction entre l'attaquant et la cible, vous verrez que nous n'aurons presque rien à écrire.

Comment doit se passer la phase de traitement ? Avant toute chose, il faut bien vérifier que le joueur est connecté et que la variable contenant notre personnage existe et n'est pas vide, sinon nous n'iront pas bien loin.
Seconde vérification : il faut demander à notre manager si le personnage que l'on veut frapper existe bien.
Si ces deux conditions sont vérifiées, alors on peut lancer l'attaque. Pour lancer l'attaque, il va falloir récupérer le personnage à frapper grâce à notre manager. Ensuite, il suffira d'invoquer la méthode permettant de frapper le personnage.

Cependant, nous n'allons pas nous arrêter là. N'oubliez pas que cette méthode peut retourner 3 valeurs différentes :

- Personnage::CEST_MOI. Le personnage a voulu se frapper lui-même.
- Personnage::PERSONNAGE_FRAPPE. Le personnage a bien été frappé.
- Personnage::PERSONNAGE_TUE. Le personnage a été tué.

Il va donc falloir afficher un message en fonction de cette valeur retournée. Aussi, seuls 2 de ces cas nécessitent une mise à jour de la BDD : si le personnage a été frappé ou s'il a été

tué. En effet, si on a voulu se frapper soi-même, aucun des deux personnages impliqués n'a été modifié.

Et voilà, si vous avez tout réussi, vous avez un jeu opérationnel !

Power Up

Ce code est très basique, beaucoup d'améliorations sont possibles. En voici quelques unes :

- Un système de niveau. Vous pourriez très bien assigner à chaque personnage un niveau de 1 à 100. Le personnage bénéficierait aussi d'une expérience allant de 0 à 100. Lorsque l'expérience atteint 100, le personnage passe au niveau suivant. Indice : le niveau et l'expérience deviendraient des caractéristiques du personnage, donc. . . Pas besoin de vous le dire, je suis sûr que vous savez ce que ça signifie !
- Un système de force. La force du personnage pourrait augmenter en fonction de son niveau, et les dégâts infligés à la victime seront donc plus importants. Indice : de même, la force du personnage serait aussi une caractéristique du personnage.
- Un système de limitation. En effet, un personnage peut en frapper autant qu'il veut dans un laps de temps indéfini. Pourquoi ne pas le limiter à 3 coups par jour ? Indice : il faudrait que vous stockiez le nombre de coups portés par le personnage, ainsi que la date du dernier coup porté. Cela ferait donc deux nouveaux champs en BDD, et deux nouvelles caractéristiques pour le personnage !
- Un système de retrait de dégâts. Chaque jour, si l'utilisateur se connecte, il pourrait voir ses dégâts se soustraire de 10 par exemple. Indice : il faudrait stocker la date de dernière connexion. À chaque connexion, vous regarderiez cette date. Si elle est inférieure à 24h, alors vous ne feriez rien. Sinon, vous retireriez 10 de dégâts au personnage puis mettriez à jour cette date de dernière connexion. Et la liste peut être longue ! Je vous encourage vivement à essayer d'implémenter ces fonctionnalités et à laisser libre court à votre imagination, vous progresserez bien plus.

Et la liste peut être longue ! Je vous encourage vivement à essayer d'implémenter ces fonctionnalités et à laisser libre court à votre imagination, vous progresserez bien plus.