Doc sur le projet de jeu php 3il

Ce cours s'adresse aux étudiants cpi2 de 3il

View on GitHub

Avant de commencer, vous devez avoir :

- Créer une base de donnée du nom de project3il

Etape 1

Objectif : Nous allons initialisé notre project, créer notre object et table character, et faire en sorte que nous puisse en ajouter via un formulaire.

Nous allons créer un répertoire class où seront placés nos fichiers de classes.

A la racine, nous aurons un fichier `header.php` qui va contenir des fonctions et notre connexion à la base puis un `index.php`

Dans notre dossier **class** nous allons créer notre premier object qui va se nommer Character.php

Cette object va représenter notre personnage avec ces différents attribut (id, name, hp, ap, password)

hp et ap seront respectivement les Point de vie et les Point d'action

Nous crérons les getter et les setter pour accéder à ces différentes variable.

Votre object doit ressembler à ça :

```
<?php
class Character
{

    private $id;

    private $name;

    private $hp;

    private $ap;
```

```
    private $password;

    public function getId()
    {
        return $this->id;
    }

    public function setId($id)
    {
        $this->id = $id;
    }

    public function getName()
    {
        return $this->name;
    }

     public function setName($name)
    {
        $this->name = $name;
    }

    public function getHp()
    {
        return $this->hp;
    }

    public function setHp($hp)
    {
        $this->hp = $hp;
    }

    public function getAp()
    {
        return $this->ap;
    }

    public function setAp($ap)
    {
        $this->ap = $ap;
    }

    public function getPassword()
    {
        return $this->password;
    }

    public function setPassword($password)
    {
        $this->password = $password;
    }
}
```

Nous rajoutons une fonction hydrate qui va nous permettra de passer un tableau en parametre pour lui donner les valeurs souhaitez (l'hydrater)

```
public function hydrate(array $donnees)
    {
        foreach ($donnees as $key => $value)
        {
            $method = 'set'.ucfirst($key);

            if (method_exists($this, $method))
            {
                $this->$method($value);
            }
        }
    }
```

Par exemple si nous donnons le tableau suivant ['name' => 'Olivier', 'hp' => 100, 'ap' => 10], cette fonction va rajouter le set sur les `key` et `ucfirst` (http://php.net/manual/fr/function.ucfirst.php) sur la premier lettre en majuscle.

Si jamais la personne se trompe dans les `key` du tableau, nous allons rajouter un test avec la fonction `method_exists` (http://php.net/manual/fr/function.method-exists.php)

Dans notre construct nous rajoutons l'appelle à cette fonction si jamais on lui passe un tableau de données :

```
    public function __construct(array $arrayOfValues = null)
    {
        if ($arrayOfValues !== null) {
            $this->hydrate($arrayOfValues);
        }
    }
```

La fonction `__construct` (http://php.net/manual/fr/language.oop5.decon.php) dans notre object permet de déclancher une action lors que nous instantion l'object par exemple : `$character = new Character();`

Ici, j'ai deux possibilités : soit je lui passe un tableau, soit je ne lui passe rien.

Si jamais je lui passe un tableau je vais lancer la fonction hydrate dessus.

Exemple : `$character = new Character(['name' => 'Olivier', 'hp' => '100', 'ap' => '10']);`

Mon object `$character` a maintenant les champs name, hp, ap d'hydrater, en faisant un echo `$character->getName()` , j'obtiens le resultat Olivier

Voila pour notre permier object.

Passons à nos deux autres fichiers qui sont vides : `index.php` et `header.php`

Pour avoir accès à notre object Character, il faut que je l'intégre dans notre header avec la fonction `require` (http://php.net/manual/fr/function.require.php) pour pouvoir l'utiliser plus tard.

Exemple :

```php
<?php
require __DIR__.'/class/Character.php';

$character = new Character(['name' => 'Olivier', 'hp' => '100', 'ap' => '10']);

echo "nom du joueur : " . $character->getName();
```

Cela fonctionne mais imaginons que dans mon répertoire class, j'ai 40 classes à charger. Ca va vite devenir relou.

Du coup, php nous propose une solution via son `spl_autoload_register`
http://php.net/manual/fr/function.spl-autoload-register.php

Ce qui nous donne ça :

```php
<?php
function loadClass($classname)
{
    require 'class//'.$classname.'.php';
}

spl_autoload_register('loadClass');
```

Ici nous allons le placer dans header pour detecter notre appel à une classe dans notre fichier, php ira la chercher en automatique.

Super ! On va tester directement ça dans notre index.php

```php
<?php
require __DIR__.'/header.php';

$character = new Character(['name' => 'Olivier', 'hp' => '100', 'ap' => '10']);

echo "Le nom du joueur est " . $character->getName() . " et à " . $character->getHp() . "
?>
```

Maintenant il faut pouvoir rentrer ce personnage en base de donnée avec PDO

On va rajouter la connexion dans notre header.php

```
$db = new PDO('mysql:host=localhost;dbname=project3il', 'root', '');
$db->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_WARNING);
```

Pensez à configurer votre dbname, le user et le mot de passe de votre environnement de dev.

On check vite fait sur la page d'index (en lancant votre navigateur tout simplement) si ça ne me fait pas de message d'erreur.

Une fois cette vérification effectué, nous allons faire un formulaire d'inscription pour créer nos personnages.

On va créer la table : characters

Penser bien au S car character est un mot qui est reservé par mysql/mariadb, si vous ne le mettez pas vous allez avoir un message d'erreur.

Dedans nous allons mettre les champs id, name, password, hp, ap.

Configurer bien votre `id` en auto incrément, name et password seront en varchar puis hp et ap en int.

on va créer une page inscription.php.

Première chose à faire vous aller mettre le require de header :

```
<?php
require __DIR__.'/header.php';
```

puis nous allons créer un formulaire en php.

Nous avons besoin que du nom du personnage et du password.

```
<form method='post'>
    <label>Nom</label>
    <input type="text" name="name">
    <label>Password</label>
    <input type="password" name="password">
    <button type="submit">Inscription</button>
</form>
```

Puis nous allons passer au traitement des données :

```php
if (isset($_POST['name']) && isset($_POST['password'])) {

    $reponse = $base->prepare('INSERT INTO characters (name, password, hp, ap) VALUES(:na
    $reponse->execute(
        [
            'name' => $_POST['name'],
            'password' => password_hash($_POST['password'], PASSWORD_ARGON2I),
            'hp' => 100,
            'ap' => 10
        ]
    );
}
```

Ici nous insérons en base notre nouveau personnage. Normalement vous devais retrouver avec une nouvelle insertion dans votre table.

C'est bien mais qu'est ce qu'il va arriver si on a deux fois le même nom ? Si on veut inserer un personnage ailleurs ?

Vous avez compris, nous allons créer une classe qui va gérer tout ça.

Nous créons un nouveau fichier dans le dossier class qui va s'appeler `CharacterRepository.php`

Ici nous allons gérer tout ce qui fait le liens entre la base de données et l'object Character

```php
<?php
class CharacterRepository
{
    private $base;

    public function __construct(PDO $base)
    {
        $this->base = $base;
    }

}
```

Nous lui passerons l'object PDO quand nous allons instancier l'object

exemple : `$characterRepository = new CharacterRepository($base);`

`$base` étant notre object PDO qui se trouve dans notre header.

Puis nous allons rajouter une fonction add, dans celle ci nous allons lui passer l'object Character

```php
public function add(Character $character)
    {
```

```
        $response = $this->base->prepare('INSERT INTO characters (name, password, hp, ap)
        $response->bindValue(':name', $character->getName());
        $response->bindValue(':password', $character->getPassword());
        $response->bindValue(':hp', $character->getHp());
        $response->bindValue(':ap', $character->getAp());

        $response->execute();

        $character->hydrate(['id' => $this->base->lastInsertId()]);
    }
```

Le `$this->base` conrrespond a notre object PDO passer dans le `__construct` pour rappel.

Nous faisons exactement la même chose que dans le traitement des données que nous avons fait dans inscription.php SAUF que cette fois on utilise les bindValue plutôt que de passer un tableau dans execute, c'est plus propre.

Puis avec la fonction de PDO `lastInsertId` , on hydrate notre object Character de son id enregistré en base.

Nous retournons dans notre fichier inscription et remplacons notre traitement de données par notre classe :

```
if (isset($_POST['name']) && isset($_POST['password'])) {

    $character = new Character([
            'name' => $_POST['name'],
            'password' => password_hash($_POST['password'], PASSWORD_ARGON2I),
            'hp' => '100',
            'ap' => '10'
    ]);

    $characterRepository = new CharacterRepository($base);
    $characterRepository->add($character);
}
```

PS : Ici, il va avoir une erreur, elle vient de la déclaration PDO plus haut, qui n'est pas nommé de la même manière.

La différence avec notre précédant traitement est que nous instantions un object Character. Nous l'hydratons avec les données du formulaire. Puis nous le rajoutons en base avec la fonction `add` de notre CharacterRepository.

C'est super !!!! mais ça ne résoud pas notre problème de doublons de nom.

Rien de plus simple, nous allons créer une nouvelle fonction dans notre repository pour checker tout ça.

```
    public function exists(Character $character)
    {
        $response = $this->base->prepare('SELECT COUNT(*) FROM characters WHERE name = :n
        $response->bindValue(':name', $character->getName());
        $response->execute();

        return (bool) $response->fetchColumn();
    }
```

Ici nous allons coté en bdd voir si il y des personnages avec le même nom.

Le type boolean (http://php.net/manual/fr/language.types.boolean.php) va nous permettre de que notre fonction renvoie true ou false suivant si il y a un résultat dans la base ou non.

```
    if ($characterRepository->exists($character) === false) {
        $characterRepository->add($character);
        echo "Votre personnage est bien créé";
    } else {
        echo "Un personnage du même nom existe";
    }
```

Nous allons remplacer dans notre fichier `inscription.php` la fonction `add` par ce code ci dessus pour rajouter la vérification et un retour de son action.

Voici notre etape 1 terminé.

Etape 2

Objectif : Nous allons pouvoir nous connecter avec notre personnage, nous allons mettre une menu dans l'etape 3 pour naviger sur ces différents fichiers

Pour ce faire nous allons créer une nouvelle page connexion.php dedans nous allons mettre un formulaire qui nous permettra de nous connecter

Notre formulaire sera encore une fois très simple :

```
<form method='post'>
    <label>Nom</label>
    <input type="text" name="name">
```

```
    <label>Password</label>
    <input type="password" name="password">
    <button type="submit">Connexion</button>
</form>
```

Puis nous allons faire le traitement des données

```
if (isset($_POST['name']) && isset($_POST['password'])) {

    $characterRepository = new CharacterRepository($base);
    if ($characterRepository->login($_POST['name'], $_POST['password'])) {
        echo "Vous etes connecter";
    } else {
        echo "Ce personnage n'existe pas";
    }
}
```

Du coup nous allons essentiellement travailler dans le CharacterRepository

Nous allons analyser nos besoins :

- Une fonction login qui va me servir à comparer les mots de passe pour me connecter pour determiner si je remplis la `$_SESSION`
- Une autre fonction pour chercher dans la base de donnéer le personnage via le nom
- Et enfin un fonction qui va me permettre de chercher un personnage via son id et qui hydrate l'object Character pour me faire une retour

Nous allons commencer par faire la fonction qui cherche le nom du personnage dans la base de donnée, elle sera presque pareil que exist :

```
    public function findByName(string $name)
    {
        $response = $this->base->prepare('SELECT * FROM characters WHERE name = :name');
        $response->bindValue(':name', $name);
        $response->execute();

        return $response->fetch();
    }
```

Très bien maintenant je peux commencer à faire la fonction login

```
    public function login(string $name, string $password)
    {
        if ($result = $this->findByName($name)) {
            if (password_verify($password, $result['password'])) {
                                return true;
```

```
            }
            return false;
        }
        return false;

    }
```

Ici je test le resultat de `findByName` pour voir si il existe puis si c'est le cas je fais un `password_verify` http://php.net/manual/fr/function.password-verify.php

Si jamais il ne trouve pas le nom ou bien le mot de passe ne correspond pas je retourne false.

Mais ce que j'aimerais c'est qu'au lieu qu'il me retourne true, il me retourne l'oject du personnage hydrater ainsi que remplir mes sessions

Pour ce faire je vais creer une nouvelle fonction `find()` dans mon CharacterRepository

```
public function find(int $id)
    {
        $response = $this->base->prepare('SELECT * FROM characters WHERE id = :id');
        $response->bindValue(':id', $id);
        $result = $response->execute();
        if ($result === true) {
            $character = new Character($response->fetch());
            return $character;
        }

        return false;

    }
```

Je cherche un personnage par rapport à son id, et si il existe je lui retourne l'oject hydrater

Du coup je modifie ma fonction `login()`

```
    public function login(string $name, string $password)
    {
        if ($result = $this->findByName($name)) {
            if (password_verify($password, $result['password'])) {
                $character = $this->find($result['id']);
                $_SESSION['id'] = $character->getId();
                $_SESSION['username'] =  $character->getName();
                return $character;
            }
            return false;
        }
        return false;

    }
```

Vu que j'utilise la variable `$_SESSION` **JE N OUBLIE SURTOUT PAS** de faire un session_start() http://php.net/manual/fr/function.session-start.php dans mon header.php

Cette fonction renvoie soit false, soit l'object du personnage.

Une fois connecté il ne devrait plus avoir accès au formulaire, je rajoute donc une protection dans mon `connexion.php`

```
if (isset($_SESSION['id'])) {
    echo "Vous êtes déjà connecter";
} else {
    ?>
    <form method='post'>
        <label>Nom</label>
        <input type="text" name="name">
        <label>Password</label>
        <input type="password" name="password">
        <button type="submit">Connexion</button>
    </form>
    <?php
}
?>
```

La connexion c'est bien mais il va falloir nous créer un espace de deconnection.

Rien de plus simple nous allons créer un fichier deconnection.php

```
<?php
require __DIR__.'/header.php';

session_destroy();
header('Location: index.php');
exit();
```

Et voila, nous avons fini l'étape deux

#Etape 3

Objectif : Nous allons mettre du html et css dans notre project avec un menu pour faciliter la navigation.

Notre `menu.php` va ressembler à ça

```
<nav class="menu">
<a href="index.php">Index</a>
```

```
<?php
if (isset($_SESSION['id'])) :?>
    <a href="deconnection.php">Déconnection</a>
<?php else: ?>
    <a href="inscription.php">Inscription</a>
    <a href="connexion.php">Connexion</a>
<?php endif ?>

</nav>
```

Nous allons rajouter dans notre header.php le doctype ainsi que le link sur notre css, ainsi que la liaison avec notre menu

```
    <!DOCTYPE html>
    <html>
    <head>
        <meta charset="utf-8">
        <link rel="stylesheet" href="style.css" >
        <title>Mon jeu</title>
    </head>
    <body>

<?php
include __DIR__.'/menu.php';
?>
```

Nous allons rajouter deux fichiers footer.php et style.css

dans le footer il n'y aura pas grand chose

```
</body>
</html>
```

et dans notre css juste le style de notre menu

```
.menu {
    width:100%;
}
```

Dans notre index on va rajouter la fiche du joueur

```
<?php
require __DIR__.'/header.php';


if (isset($_SESSION['id'])) {
    $characterRepository = new CharacterRepository($base);
```

```
    $character = $characterRepository->find($_SESSION['id']);
    ?>
    <table>
        <tr>
            <th>Nom du joueur</th>
            <td><?= $character->getName(); ?></td>
        </tr>
        <tr>
            <th>Point de vie</th>
            <td><?= $character->getHp(); ?></td>
        </tr>
        <tr>
            <th>Point d'action</th>
            <td><?= $character->getAp(); ?></td>
        </tr>
    </table>
    <?php
}

require __DIR__.'/footer.php';

?>
```

#Etape 4

Objectif : Nous allons permettre les combats entre deux personnages

Dans un premier temps nous allons afficher tout les combatants disponibles.

Il y avait une erreur dans l'étape 1, vérifier votre __construct dans votre character.

```
    public function __construct(array $arrayOfValues = null)
    {
        if ($arrayOfValues !== null) {
            $this->hydrate($arrayOfValues);
        }
    }
```

Dans notre Repository, nous créons une nouvelle fonction

```
    public function findAllWithoutMe(int $id)
    {
        $response = $this->base->prepare('SELECT * FROM characters WHERE id <> :id');
        $response->bindValue(':id', $id);
        $result = $response->execute();
        if ($result === true) {
            $records = $response->fetchAll(PDO::FETCH_CLASS, 'Character');
```

```
            return $records;
        }

        return false;
    }
```

avec PDO::FETCH_CLASS, nous allons pouvoir hydrater directement notre object sans lui passer de tableau c'est PDO qui va s'occuper de remplir les getter et les setter (http://php.net/manual/fr/pdostatement.fetchall.php). Bien sur nous nous excluons du résultat.

Sur notre index, dans une premier temps, nous allons faire le listing des personnes autour de nous.

```
    $listOfCharacter = $characterRepository->findAllWithoutMe($_SESSION['id']);
    foreach ($listOfCharacter as $character):?>
        <a href="attaque.php?id=<?= $character->getId();?>"><?= $character->getName();?><
    <?php endforeach;
```

Pour attaquer nous allons devoir récupérer `$id` de l'attaquant que nous transmettrons à notre nouvelle page `attaque.php` sous forme dans liens a où nous allons traiter le combat. Nous allons passer l'id de l'adversaire en paramêtre pour le récupérer avec `$_GET`

Notre fichier attaque maintenant :

```
<?php
require __DIR__.'/header.php';

if (isset($_SESSION['id'])) {

    $characterRepository = new CharacterRepository($base);
    $myCharacter = $characterRepository->find($_SESSION['id']);
    $enemy = $characterRepository->find($_GET['id']);

    echo $myCharacter->getName() . " attaque ". $enemy->getName();
}

require __DIR__.'/footer.php';
?>
```

Ici on récupère deux objects `Character` et nous affichons juste qui attaque qui.

Pour les attaques, nous allons créer une fonction dans le CharacterRepository pour permettre la mise à jour les points de vie

```
public function updateHp(Character $character)
    {
```

```
        $response = $this->base->prepare('UPDATE characters SET hp = :hp WHERE id = :id')

        $response->bindValue(':hp', $character->getHp(), PDO::PARAM_INT);
        $response->bindValue(':id', $character->getId(), PDO::PARAM_INT);

        $response->execute();
    }
```

Puis dans notre fichier attaque.php nous allons faire le combat

```
    $damage = rand(1,100);
    $hp = $enemy->getHp() - $damage;
    $enemy->setHp($hp);
    $die = false;
    if ($hp < 0) {
        $die = true;
    }
    $characterRepository->updateHp($enemy);

    echo $myCharacter->getName() . " attaque ". $enemy->getName(). " pour " . $damage ."
    if ($die === true) {
        echo $enemy->getName(). " est mort";
    }
```

Nous allons utiliser une valeur aléatoire avec la fonction rand (http://php.net/manual/fr/function.rand.php)

Si jamais c'est point de vie sont en dessous de zéro nous allons le concidérer comme mort.

On va donc rajouter un champs dans notre object pour savoir son "state", voir si il est en vie ou mort.

Dans notre Character.php nous allons rajouter des constantes (http://php.net/manual/fr/language.constants.php) au dessus de nos variables

```
    public const ALIVE = 'alive';
    public const DEAD = 'dead';
```

puis une fonction

```
    public function getState()
    {
        if ($this->hp < 0) {
            return self::DEAD;
        }
```

```
        return self::ALIVE;
    }
```

Du coup on va faire une amélioration sur notre fonction d'attaque

```
if ($enemy->getState() === Character::DEAD) {
        echo "Vous venez de découvrir un corp sans vie ...";
    } else {
        $damage = rand(1,100);
        $hp = $enemy->getHp() - $damage;
        $enemy->setHp($hp);

        $characterRepository->updateHp($enemy);

        echo $myCharacter->getName() . " attaque ". $enemy->getName(). " pour " . $damage
        if ($enemy->getState() === Character::DEAD) {
            echo $enemy->getName(). " est mort";
        }
    }
```

Maintenant que l'attaque est faite, nous allons ajouter le fait de devoir dépenser des points d'actions pour attaquer.

Nous allons définir son cout dans une constante dans notre object personnage

```
    public const ATTAQUE_COST = 5;
```

Puis dans notre attaque nous allons rajouter cette condition

```
if ($enemy->getState() === Character::DEAD) {
        echo "Vous venez de découvrir un corp sans vie ...";
    } else {
        if ($myCharacter->getAp() >= Character::ATTAQUE_COST) {

            // Point d'action
            $myCharacter->setAp($myCharacter->getAp() - Character::ATTAQUE_COST);
            $characterRepository->updateAp($myCharacter);

            // Attaque
            $damage = rand(1,100);
            $hp = $enemy->getHp() - $damage;
            $enemy->setHp($hp);
            $characterRepository->updateHp($enemy);

            echo $myCharacter->getName() . " attaque ". $enemy->getName(). " pour " . $da
            if ($enemy->getState() === Character::DEAD) {
                echo $enemy->getName(). " est mort";
```

```
            }
        } else {
            echo "Vous n'avez pas assez de point d'action";
        }
    }
```

Nous testons si le personnage a assez de point d'action pour attaquer, si il en a assez nous continuons l'action tout en mettant à jour les points actions du personnage.

Il faudra créer la fonction updateAp qui est assez simple du coup je vous laisse faire.

Et voici notre etape 4 finalisé

#Etape 5

Objectif : Regénération des points d'action

Pour recuperer des points d'action nous allons intéragir avec des dates, et en php on manipule les dates avec l'object Datetime (http://php.net/manual/fr/class.datetime.php). Bonne lecture de la doc, vous en aurez besoin.

Alors nous allons rajouter quelques variables à notre object de personnage

Pour commencer on va définir à combien est la régénération de point d'action et combien de point d'action maximun on peut avoir.

Je vais définir deux constantes :

```
    public const AP_REGEN = 60;
    public const AP_MAX = 20;
```

On va calculer ces points d'action par rapport à une date que l'on va nommer `$lastaction`

Vous allez la rajouter dans votre class avec son getter & setter associer puis dans votre base de données en DATETIME

Une fois fait cela, on va définir une fonction qui va permettre de définir le nombre de point action à donner au moment de la connexion.

Dans notre object on va donc comparer les object de dates dans notre `Character.php`

```
public function getNewAp()
    {
        $datetime1 = new DateTime('now');
        $datetime2 = new DateTime($this->lastaction);
```

```
        $interval = $datetime1->diff($datetime2);
        $seconde = $interval->s + $interval->i * 60 + $interval->h * 60 * 60;
        if ($seconde > self::AP_REGEN) {
            $newAP = floor($seconde / self::AP_REGEN);
            $this->ap = $this->ap + $newAP;
        }
    }
```

On calcul la différence entre la date de dernier connexion qui est dans la base et la date de l'instant.

On regarde si les secondes de différents font plus que les secondes que nous avons mise en constantes (60 s = 1 point d'action je le rappelle)

Et si c'est le cas, on calcul le nombre de point d'action a rajouté au personnage.

On utilise différentes nouveautés que vous retrouverez dans la doc (http://php.net/manual/en/datetime.diff.php, http://php.net/manual/en/function.floor.php)

Du coup on implémente cette fonction au niveau du login dans notre `CharacterRepository`

```
if ($result = $this->findByName($name)) {
            if (password_verify($password, $result['password'])) {
                $character = $this->find($result['id']);
                $_SESSION['id'] = $character->getId();
                $_SESSION['username'] =  $character->getName();
                $character->getNewAp();
                $this->updateLastActionAndAp($character);
                return $character;
            }
            return false;
        }
        return false;
```

Et bien sur il faudra mettre à jour notre personnage d'où la fonction `updateLastActionAndAp` qu'il faudra créer. Cette fonction servira à mettre à jour la date de connexion dans la variable `lastaction` et `ap` .

Exemple :

```
public function updateLastActionAndAp(Character $character)
    {
        $datenow = new DateTime('now');
        $response = $this->base->prepare('UPDATE characters SET lastaction = :lastaction,
        $response->bindValue(':lastaction', $datenow->format('Y-m-d H:i:s'), PDO::PARAM_S
        $response->bindValue(':ap', $character->getAp(), PDO::PARAM_INT);
        $response->bindValue(':id', $character->getId(), PDO::PARAM_INT);
```

```
        $response->execute();
    }
```

Pour avoir en permance mon personnage une fois connecter, dans mon header.php je rajoute

```
if (isset($_SESSION['id'])) {
    $characterRepository = new CharacterRepository($base);
    $character = $characterRepository->find($_SESSION['id']);
}
```

et dans mon menu.php

```
    <?php if (isset($_SESSION['id'])) : ?>
        <div>
            PA : <?= $character->getHp(); ?>, AP : <?= $character->getAp(); ?>
        </div>
    <?php endif ?>
```

J'ai maintenant tout le temps mes points de vie et mes points d'action d'afficher.

Je viens de finir l'étape 5 et mon jeu est fonctionnel

#Etape 6

Objectif : Création d'un journal qui liste les différentes actions

Pour commencer nous allons créer l'object `CharacterLog.php`

Dedans il y aura quatre variables : id, message, add_at, character_id

Une fois fait, créer la table dans votre base de données.

Nous allons créer son pendant Repository :

```
<?php

class CharacterLogRepository
{
    private $base;

    public function __construct(PDO $base)
    {
        $this->base = $base;
    }
```

```
    public function add(Character $character, $message)
    {
        $datenow = new DateTime('now');

        $response = $this->base->prepare('INSERT INTO characters_log (message, add_at, ch
        $response->bindValue(':message', $message);
        $response->bindValue(':add_at', $datenow->format('Y-m-d H:i:s'), PDO::PARAM_STR);
        $response->bindValue(':character_id', $character->getId());

        $response->execute();

    }

    public function findAllForMe(int $id)
    {
        $response = $this->base->prepare('SELECT * FROM characters_log WHERE character_id
        $response->bindValue(':id', $id);
        $result = $response->execute();
        if ($result === true) {
            $records = $response->fetchAll(PDO::FETCH_CLASS, 'CharacterLog');
            return $records;
        }

        return false;
    }

}
```

Nous avons deux fonctions, une pour ecrire dans le journal, l'autre pour afficher sur le journal du personnage.

On va passer l'écrire du journal dans le fichier `attaque.php`

On avait déjà le message de fait, il suffira de le mettre en variable et de le passer à notre fonction `add`

```
            // J'enregistre les logs dans chaques journal
            $characterLogRepository = new CharacterLogRepository($base);
            $characterLogRepository->add($myCharacter, $message);
            $characterLogRepository->add($enemy, $message);
```

J'enregistre dans mon journal et dans celui que j'attaque pour qu'il est une trace.

Puis je créer un fichier `journal.php` ou je vais afficher les actions lié au personnage.

```
<?php
require __DIR__.'/header.php';
```

```
if (isset($_SESSION['id'])) {
    $characterLogRepository = new CharacterLogRepository($base);
    if ($listOfLog = $characterLogRepository->findAllForMe($_SESSION['id'])):
    foreach ($listOfLog as $log):?>
        <?= $log->getAddAt();?> : <?= $log->getMessage();?><br>
    <?php
    endforeach;
    endif;
}

require __DIR__.'/footer.php';

?>
```

Il faudra aussi mettre une entrée dans notre menu.php pour accéder au journal du personnage.

L'étape 6 est fini.

#Etape 7

Objectif : Création des soins

On veut commencer à diversifier notre jeu, pour ce faire nous allons implémenter le soins.

Comme pour l'attaque, il faudra définir un cout dans la class Character :

```
    public const HEAL_COST = 2;
```

Nous allons lui donner 2, puis nous allons créer le fichier `heal.php` , il aura pratiquement le même déroulement que `attaque.php`

```
<?php
require __DIR__.'/header.php';

if (isset($_SESSION['id'])) {
    $characterRepository = new CharacterRepository($base);
    $myCharacter = $characterRepository->find($_SESSION['id']);
    $friend = $characterRepository->find($_GET['id']);

    if ($friend->getState() === Character::DEAD) {
        echo "Vous venez de découvrir un corps sans vie ...";
    } else {
        if ($myCharacter->getAp() >= Character::HEAL_COST) {

            // Point d'action
            $myCharacter->setAp($myCharacter->getAp() - Character::HEAL_COST);
```

```
            $characterRepository->updateAp($myCharacter);

            // Heal
            $heal = rand(1,50);
            $hp = $friend->getHp() + $heal;
            $friend->setHp($hp);
            $characterRepository->updateHp($friend);

            $message = $myCharacter->getName() . " soigne ". $friend->getName(). " pour "

            echo $message;

            // J'enregistre les logs dans chaques journal
            $characterLogRepository = new CharacterLogRepository($base);
            $characterLogRepository->add($myCharacter, $message);
            $characterLogRepository->add($friend, $message);

        } else {
            echo "Vous n'avez pas assez de point d'action";
        }
    }
}

require __DIR__.'/footer.php';

?>
```

Tout est pret, il ne reste plus qu'a donné la possiblité à l'utilisateur d'utiliser l'un ou l'autre, ça va se passer dans `index.php`

```
        <?= $character->getName();?> : Action disponible <a href="attaque.php?id=<?= $cha
```

En peu de temps nous avons diversifié nos actions dans le jeu, on peut heal ou attaquer.

#Etape 8

Objectif : On mets en place les hp max, les ap max ainsi que du leveling

On va faire les constantes pour les max

```
    public const HP_MAX = 100;

    public const AP_MAX = 100;
```

Du coup on va faire un refactor des deux fonctions qui mette en place les HP et AP

Pour les AP ça va être dans un la fonction `getNewAp()`

```
            if ($this->ap > self::AP_MAX) {
                $this->ap = self::AP_MAX;
            }
```

et pour les HP ça va être dans setHp tout simplement, on va du coup créer aussi la fonction getHpMax pour prendre en compte les levels que l'on va implémenter.

```
    public function setHp($hp)
    {
        if ($hp > $this->getHpMax()) {
            $this->hp = $this->getHpMax();
        } else {
            $this->hp = $hp;
        }
    }

    public function getHpMax()
    {
        return self::HP_MAX;
    }
```

On va maintenant rajouter des deux variables dans notre object `Character.php` qui vont être experience et level, il faudra aussi les rajouter dans votre table

On pourra leur donner aussi des valeurs par defaut a ces deux varibles de cette manière :

```
    private $experience = 0;

    private $level = 1;
```

Du coup j'en profite pour faire une petit refactor sur le repository en créer une fonction update plus générique

```
    public function update(Character $character)
    {
        $character->checkExperience();
        $response = $this->base->prepare('UPDATE characters SET hp = :hp, ap = :ap, exper
        $response->bindValue(':ap', $character->getAp(), PDO::PARAM_INT);
        $response->bindValue(':experience', $character->getExperience(), PDO::PARAM_INT);
        $response->bindValue(':hp', $character->getHp(), PDO::PARAM_INT);
        $response->bindValue(':level', $character->getLevel(), PDO::PARAM_INT);
        $response->bindValue(':id', $character->getId(), PDO::PARAM_INT);
```

```
        $response->execute();
    }
```

On va créer une fonction checkExperience pour faire passer les levels, on va créer une constante du palier d'expérience dans la classe Character que l'on va mettre à 1000

```
    public function checkExperience()
    {
        $experienceMax = $this->level * self::LEVEL_EXPERIENCE;
        if ($this->experience >= $experienceMax) {
            ++$this->level;
            $this->experience = 0;
        }
    }
```

Il faudra donc rajouter un ajout d'expérience à chaque attaque ou chaque soins ET rajouter un ajout bonus si la personne le tue.

Je vous laisse mettre ça dans `attaque.php` et `heal.php`

Les gains sont :

Attaque	Mort	Soin
100 exp	500 exp	50 exp

Après nous rajoutons la gestion de la mort dans le `header.php`

```
if (isset($_SESSION['id'])) {
    $characterRepository = new CharacterRepository($base);
    $character = $characterRepository->find($_SESSION['id']);
    if ($character->getState() === Character::DEAD) {
        echo "Vous êtes mort mais rien n'est fini pour vous !";
        $character->setHp($character->getHpMax());
        $characterRepository->update($character);
    }
}
```

Il faudra rajouter les indications des HP max, l'expérience et le level dans le `menu.php` `

Voici la fin de l'étape 8
