académie Aix-Marseille
GRETA PROVENCE
FORMEZ-VOUS À DEMAIN

PHP 5-7

Découvrir la programmation orienté objet et la conception MVC

Connaître les bases du Framework Symfony

Maîtriser le Framework de persistance Doctrine

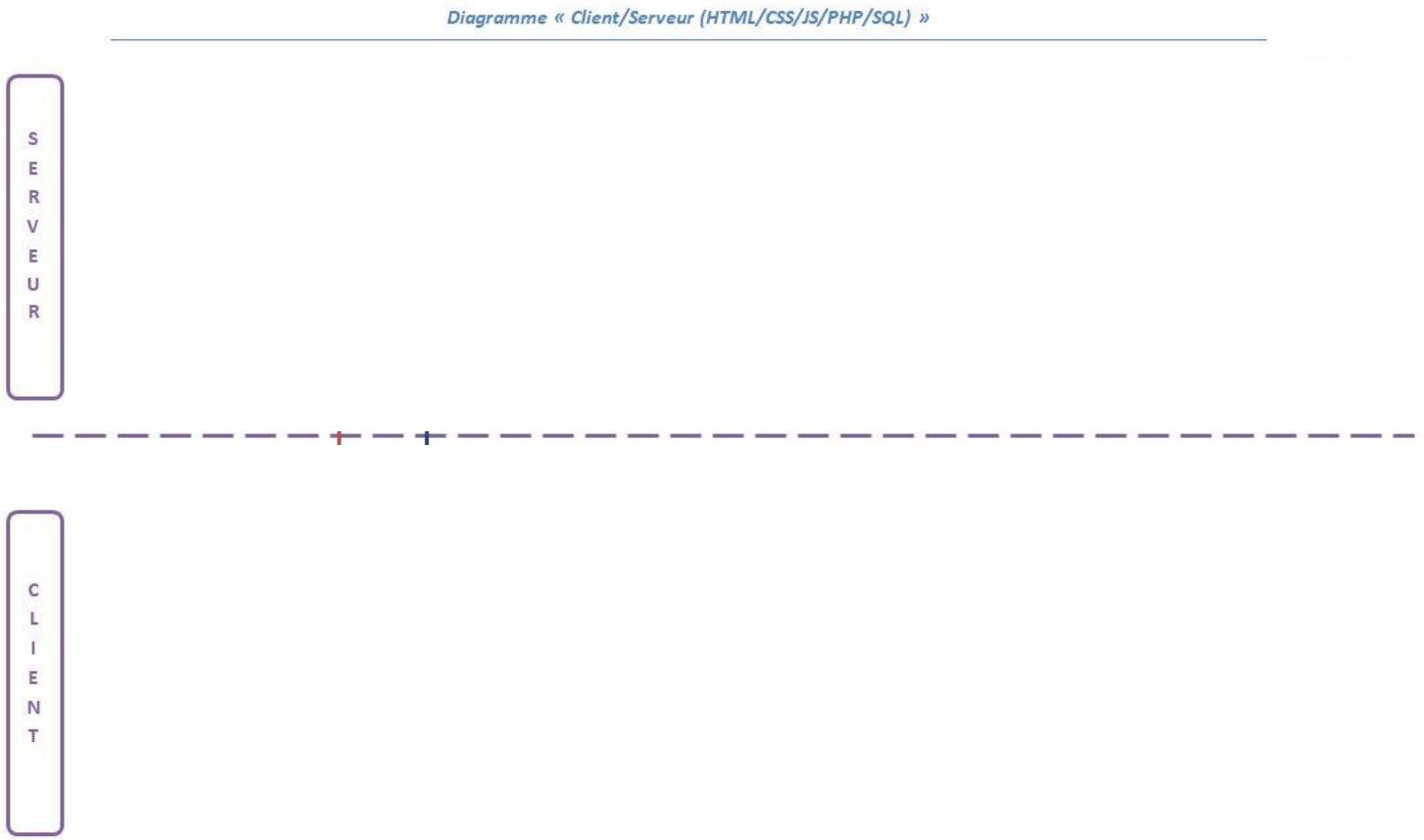
Rappels

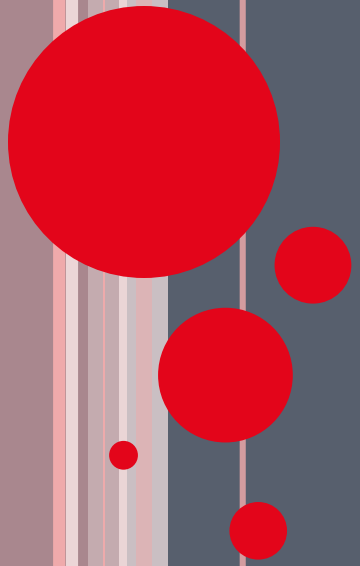
HTML5 / CSS3 / JS / jQuery / Bootstrap

PHP / SQL

Cookies et sessions

RAPPELS CLIENT-SERVEUR





PHP

Historique et rappels

De PHP4 à PHP 7.4

- PHP 4 (2000)
  - Le constructeur a le nom de la classe
  - Pas de vraie notion « objet »
  - Passages des variables par valeur

- PHP 5 (2004) - PHP 5.6 (2014)
  - Le constructeur s'appelle __construct ( )
  - Notion « objet » (visibilité des attributs, classes abstraites et finales)
  - Passages des objets par référence
  - Héritage, interfaces et traits (PHP 5.4)
  - Exceptions

De PHP4 à PHP 7.4

- PHP 7.0 (2015)
  - Amélioration des performances (2x)
  - Type de retour sur les fonctions
  - Spaceship operator (<=>)
  - Null Coalesce Operator (??)
  - Classes anonymes (function () { … })
- PHP 7.1 (2016)
  - Type de retour « void » et « null possible » (hello() : ?int)
  - Extraction de tableaux avec des crochets ([$a, $b] = $aArr)
  - Visibilité des constantes de classes

De PHP4 à PHP 7.4

- PHP 7.2 (2017)
  - Nouveau type « object »
- PHP 7.3 (2018)
  - Syntaxes heredoc/nowdoc améliorées
  - Virgule autorisée en fin de fonction
  - array_key_first( ) / array_key_last( )
- PHP 7.4 (2020)
  - Propriétés typées (public string $name)
  - Fonctions flèches (array_map(fn($n) => $n * $factor, [...]))
  - Opérateur d'assignement de fusion Null (??=)
  - Déballage dans les tableaux (['AA', 'BB', ...$aParts, 'CC'])

Les bases du langage PHP (1/2)

- Langage de programmation utilisé principalement pour produire des pages web dynamiques

- Concepts communs
  - Variables, types simples et complexes
  - Opérateurs (+ - / * % == === != !== ++ -- && || !)
  - Structures de contrôles
    - Les conditions (if/else ; switch/break)
    - Les boucles (while ; do..while ; for ; foreach)
  - Les fonctions
    - Paramètres et valeur de retour
    - Notion de « scope » (portée des variables)

Les bases du langage PHP (2/2)

- Implémentations propres à PHP
  - Tableaux
    - Parcours de tableau : foreach ($aArray as $iKey => $sVal)
    - array_key_exists (..) ; in_array (..) ; array_search (..)
  - Affichage et débogage
    - echo 'Bonjour Fabien !';
    - print_r ($variable)
  - Concaténation de variable
    - $sVar = $sStr1 . ' lorem epsum ' . $sStr2;
    - $sVar .= $sStr3;
  - Inclusion de fichiers
  - Le type NULL

Exercice 1

- Créer un programme exécutable en ligne de commande
  - Le programme doit demander le prénom de l'utilisateur et lui dire « Bonjour {Prénom} ! »

```
F2000@F2000-PC MINGW64 /d/Formation/PHP-PE
$ php exo1.php
== Début du programme ==
Quel est votre prénom ? Fabien
Bonjour Fabien !
== Fin du programme ==
```

  - readline (..) demande une saisie à l'utilisateur

PHP et la POO

Classes, objets, méthodes et propriétés
- Visibilité des attributs
- Le constructeur
- L'héritage , les interfaces et les traits

Gestion des exceptions

Les espaces de nommage

Les classes

- Objet « complexe » qui permet de représenter de entités ayant des propriétés et des comportements
  - Les propriétés sont représentées par des attributs
  - Les comportements sont représentés par des fonctions

- Notion de visibilité
  - public (*par défaut* ; accessible depuis l'extérieur)
  - protected (accessible au travers de l'héritage)
  - private (non accessible hormis au sein de la classe)

- Classe = référentiel de construction
- Instance = exemplaire unique de la classe (entité)

Les classes

Exercice 2

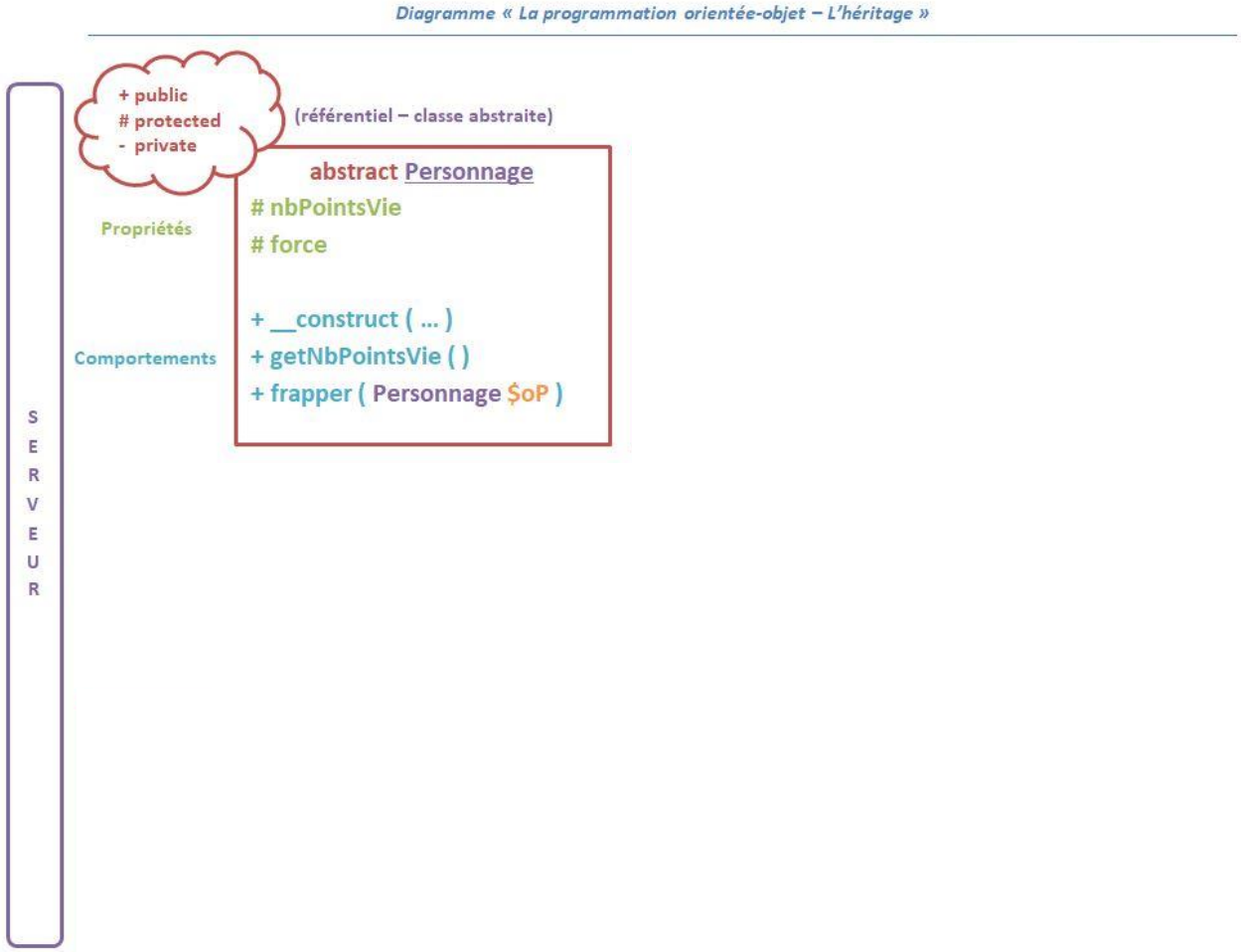
- Créer deux classes « Warrior » et « Wizard »
  - Chaque instance de Warrior doit posséder
    - Un nom, des points de vie, des points de force
  - Chaque instance de Wizard doit posséder
    - Un nom, des points de vie, des points de force, des points de magie

```
F2000@F2000-PC MINGW64 /d/Formation/PHP-PE
$ php exo2.php
== Début du programme ==
War - [H: 100] - [S: 10]
Wiz - [H: 100] - [S: 10]- [M: 10]
== Fin du programme ==
```

L'HÉRITAGE

- Permet de simplifier et d'optimiser la conception des classes
  - class Guerrier extends Personnage
- Une classe peut hériter d'une autre classe et ainsi récupérer ses propriétés et ses comportements publiques ou protégés
  - On parle alors de classe fille et de classe parente
- Classes abstraites et finales
  - abstract class
  - final class

L'HÉRITAGE



Exercice 3

- Créer la classe « Character »
  - Reprendre les points communs des classes « Warrior » et « Wizard »

```
F2000@F2000-PC MINGW64 /d/Formation/PHP-PE
$ php exo3.php
== Début du programme ==
War - [H: 100] - [S: 50]
Wiz - [H: 100] - [S: 10]- [M: 10]
== Fin du programme ==
```

Les interfaces

- Permet de déclarer des comportements à respecter
  - class Character implements DbManagerInterface

- L'implémentation des comportements n'est pas fournie par l'interface. Seul le cadre (prototype) est fourni

```
interface DbManagerInterface {
    public function loadAll () : array;
    public function get (int $iId) : object;
    public function save (object $oObject) : void;
}
```

Exercice 4

- Créer l'interface « Loggable »
  - L'interface doit permettre d'enregistrer des messages dans un fichier de log

```
F2000@F2000-PC MINGW64 /d/Formation/PHP-PE
$ php exo4.php
== Début du programme ==
War - [H: 100] - [S: 50]
Wiz - [H: 100] - [S: 10]- [M: 10]
== Fin du programme ==

F2000@F2000-PC MINGW64 /d/Formation/PHP-PE
$ cat log.txt
Je suis là (War)
Je suis là (Wiz)
```

Les traits

- Permet d'encapsuler des propriétés et/ou des comportements et de les réutiliser à volonté
  - class AbstractCharacter
    - use Counter;
- L'implémentation des comportements doit être fournie par le trait.

```
trait Counter {
        public $i;

        public function increment(int $iStep = 1) : void {
                $i += $iStep;
        };
}
```

Exercice 5

- Créer le trait « Positionable »
  - Le trait doit permettre d'enregistrer des coordonnées x/y et de les afficher

```
F2000@F2000-PC MINGW64 /d/Formation/PHP-PE
$ php exo5.php
== Début du programme ==
War - [H: 100] - [S: 50]
Wiz - [H: 100] - [S: 10]- [M: 10]
== Fin du programme ==

F2000@F2000-PC MINGW64 /d/Formation/PHP-PE
$ cat log.txt
Je suis en [1,1] (War)
Je suis en [2,5] (Wiz)
```

Les exceptions

- Permet de mieux gérer les erreurs dans un programme et de proposer des actions correctives

- throw — permet de lancer une exception. Celle-ci va remonter la pile d'exécution tant qu'elle n'est pas « attrapée »

- try — permet d'écouter des exceptions dans une section de code
- catch — permet d'attraper une exception
- finally — permet de finaliser la section de code

Exercice 6

- Créer la classe « Game »
  - La classe Game va simuler un plateau de jeu 10x10
  - La classe Game va pouvoir manipuler des objets « AbstractCharacter »
  - Elle doit émettre une exception si on essaie d'ajouter un joueur déjà présent ou si les coordonnées du jeu sont invalides

```
F2000@F2000-PC MINGW64 /d/Formation/PHP-PE
$ php exo6.php
>> Une exception est survenue (Le joueur est déjà présent (Wiz))
== Début du programme ==
War - [H: 100] - [S: 50]
Wiz - [H: 100] - [S: 10]- [M: 10]
== Fin du programme ==

F2000@F2000-PC MINGW64 /d/Formation/PHP-PE
$ php exo6.php
>> Une exception est survenue (Le joueur a des coordonnées invalides ([-1,-1]))
== Début du programme ==
War - [H: 100] - [S: 50]
Wiz - [H: 100] - [S: 10]- [M: 10]
== Fin du programme ==
```

Les namespaces (espaces de noms)

- Moyen d'encapsuler des éléments
  - Exemple d'un fichier foo.txt présent dans plusieurs répertoires
  - Très utile pour les auteurs de bibliothèques et d'applications
- Permet d'empêcher la collision de noms entre votre code et des bibliothèques tierces

Les namespaces (espaces de noms)

```php
// Model\AbstractCharacter.php
namespace Model;

class AbstractCharacter {

}

// index.php
use Model\AbstractCharacter;                    // raccourci
use Model\AbstractCharacter as AbsCha;          // alias

$a = new AbstractCharacter;  // new Model\AbstractCharacter;
$a = new AbsCha;             // new Model\AbstractCharacter;
```

Exercice 7

- Implémenter les namespaces suivants
  - App
  - App\Entity

Auto-chargement de classes

- Permet de charger automatiquement une classe selon un pattern défini

```
spl_autoload_register ( function ( $class ) {
        include_once ('class/' . $class . '.php');
} );
```

Exercice 8

- Retirer les « include (..) » et charger nos classes de manière dynamique grâce à l'autoloader et à nos espaces de noms

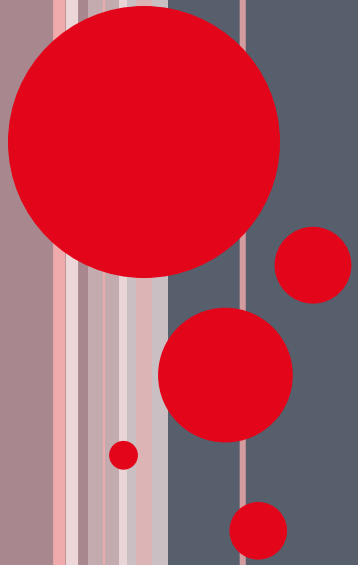
Atelier 1

- Réaliser un petit programme permettant à chaque joueur de se déplacer tour à tour

```
F2000@F2000-PC MINGW64 /d/Formation/PHP-PE
$ php index.php
== Début du programme ==
[1] Nouveau jeu
> 1

== GAME ==
Gandalf - [H: 150] - [S: 20]

Dark Vador - [H: 200] - [S: 50]

C'est à vous de jouer, Gandalf([5,4]), que voulez-vous faire ?
[M] Se déplacer
[Q] Quitter le jeu
> M
Direction (T/B/L/R)? T
C'est à vous de jouer, Dark Vador([11,4]), que voulez-vous faire ?
[M] Se déplacer
[Q] Quitter le jeu
> M
Direction (T/B/L/R)? L

== GAME ==
Gandalf - [H: 150] - [S: 20]

Dark Vador - [H: 200] - [S: 50]

C'est à vous de jouer, Gandalf([5,3]), que voulez-vous faire ?
```

Atelier 2

- Créer les classes « Snake », « Spider » et « SpiderQueen »
  - Force constante
  - Santé pouvant être réduite en cas d'attaque
- Créer les classes « Potion » et « Weapon »
  - Attributs : nom, poids, taille, valeur d'action
- Un « Character » doit pouvoir récupérer des objets au sol
- Après le tour des joueurs, les monstres se déplacent aléatoirement



Conception MVC

La couche modèle et données

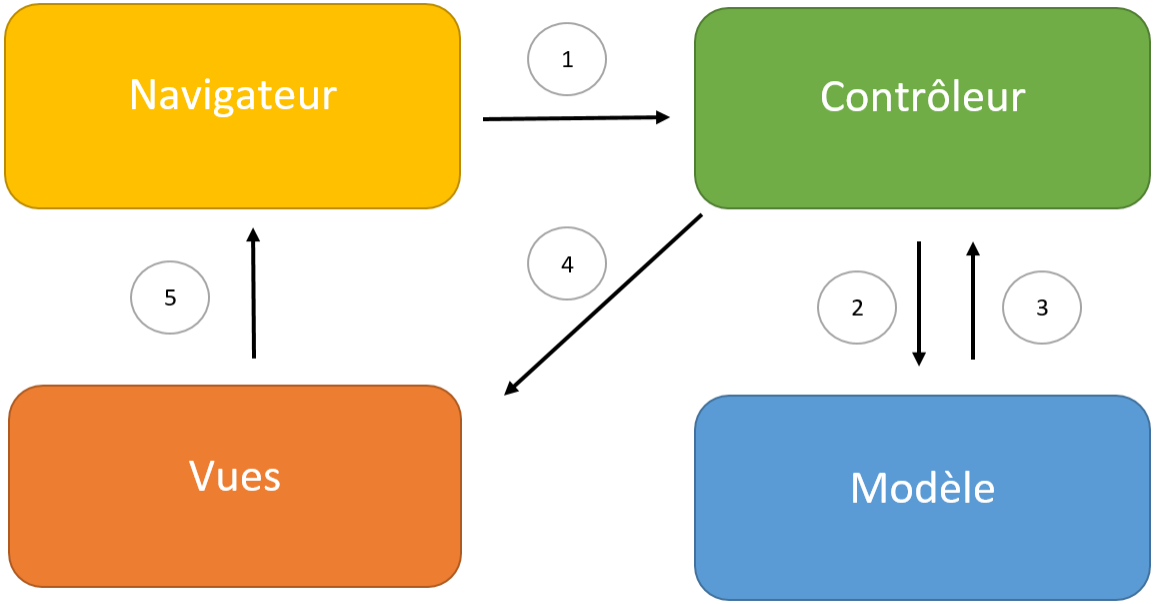
Le rôle du contrôleur

Le rôle des vues

La conception MVC

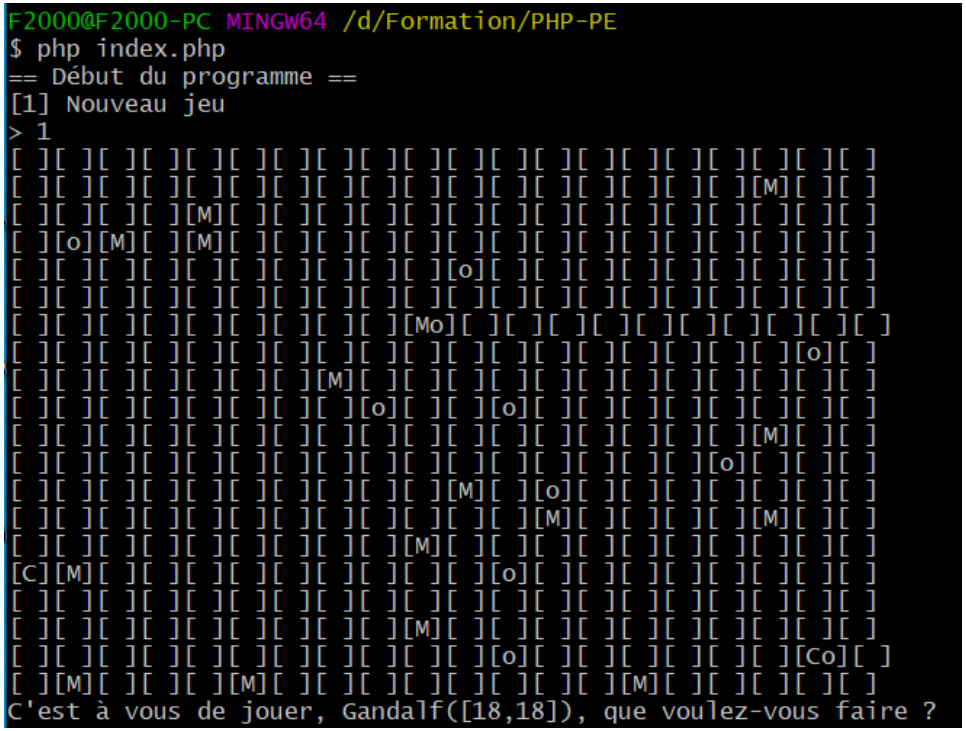
- « Design-pattern » très répandu permettant de mieux séparer le code source d'un programme
  - **Modèle** : classes métiers (Entity / Model). Représente le cœur de votre application.
  - **Vue** : représentation (graphique ou non) de votre applicatif. Cela peut être des vues HTML ou des sorties JSON. Permet d'interagir avec l'applicatif.
  - **Contrôleur** : outil de liaison entre le modèle et les vues. Point central de votre applicatif. *Dans un site internet, le contrôleur s'occupe du « routing ».*

La conception MVC



Exercice 9

- Créer une vue « board.php » qui va afficher le plateau de jeu
- Créer une classe « Model\Board » qui va s'occuper du plateau de jeu et de ses interactions



Les managers et les services

- Surcouche du « design-pattern » MVC, sur la partie « Modèle »

- Modèle « **Manager** » : classe métier regroupant des fonctionnalités liées à un cas d'utilisation « précis »
  - UserManager — Classe se chargeant de manipuler des « User » (modification du compte, etc.)
  - ExportManager — Classe se chargeant d'exporter des données

- Modèle « **Service** » : classe métier dédiée à une fonctionnalité précise
  - CsvService — Classe dédiée à la génération de fichiers CSV
  - PdfService — Classe dédiée à la génération de fichiers PDF
  - EmailService — Classe dédiée à l'envoi de mail

Exercice 10

- Créer un manager « GameManager » qui doit permettre de sauvegarder un « Game » dans un fichier texte *via la sérialisation*

- Concept de *sérialisation* : permet le stockage de données au format « texte »
  - *Transtype et linéarise un objet*
  - serialize ( ) transforme un « objet » en chaîne de caractères
  - unserialize ( ) transforme une chaîne de caractères en « objet »