TP 8 : Modèle MVC

Objectifs du TP:

- Structurer un site WEB selon le modèle MVC.

---

1. MVC : Convention de nommage

La réussite d'un projet passe avant tout par une bonne organisation. Parmi les règles que l'on peut se donner, les conventions de nommage permettent d'avoir une indication sur la signification de l'objet associé à un mot, uniquement grâce à la morphologie de ce mot. Vous pouvez voir quelques propositions de conventions de nommage en PHP :

- Convention pour les identificateurs : https://jcrozier.developpez.com/tutoriels/web/php/conventions-nommage/
- Quelques idées, un modèle MVC assez complexe : https://pub.phyks.me/sdz/sdz/votre-site-php-presque-complet-architecture-mvc-et-bonnes-pratiques.html
- Une ressource intéressante sur PHP mais 'très technique' : https://eilgin.github.io/php-the-right-way/

Nous vous proposons les conventions de nommage suivantes :

- **model/*NomDeClasse*.class.php :** le modèle ne contient que des classes. Chaque fichier ne doit contenir que la description d'une seule classe avec un nom en CamelCase.
- **model/*NomDeClasse*.test.php :** le test de la classe. Ce test ne doit pas produire de HTML et doit être lancé par la commande `php NomDeClasse.test.php`.
- **controler/*nomcontrôleur*.ctrl.php :** un contrôleur gère une série d'actions de l'interface. Il ne contient que du PHP et inclut les classes utiles du modèle. Le nom du fichier est de type lower camelCase.
- **view/*nomDeVue*.view.html :** les vues sont **les seuls fichiers** qui contiennent du code HTML. Si le suffixe est `html` alors c'est une **vue statique**, c'est-à-dire qui ne contient pas de PHP. Le nom du fichier est de type lower camelCase.
- **view/*nomDeVue*.view.php :** les **vues dynamiques** doivent avoir le suffixe php.
- **view/*nomDeVue*.test.php :** on peut tester l'affichage d'une vue avec ce fichier, qui contient des valeurs pour la vue.
- **config/config.ini :** le répertoire `config` contient les informations de configuration de l'application. Le fichier `config.ini` est le fichier de la configuration principale de l'application.
- **framework/*nomDeClasse*.fw.php** : le répertoire `framework` contient du code PHP non dépendant de l'application, comme la gestion des vues par inclusion dans une méthode.
- **helper/*nomFicher*.help.php** : le répertoire `helper` peut contenir du code d'usage général réutilisable qui vient compléter la bibliothèque standard.

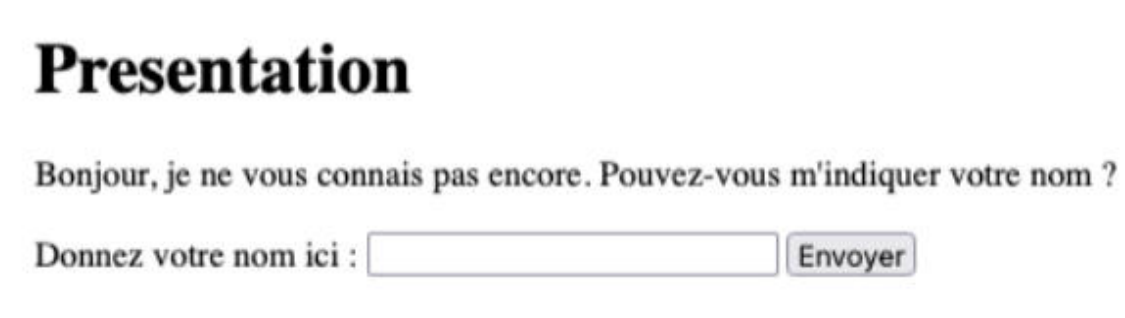
2. Un "micro" framework MVC (salutation multilingue)

Nous vous proposons de réaliser une application simple, structurée en MVC à d'aide d'un "micro" framework. Un framework est une organisation et un ensemble de codes source pour rapidement mettre en action un patron de conception. Le framework est minimal (i.e. "micro") car il est composé seulement de deux fichiers :

- **index.php** : ce fichier à la racine de l'application fait office de **routeur**. Il redirige toutes les demandes (URL) vers un fichier contrôleur. Concrétement, il ne fait que **charger un fichier** du répertoire `controler`.
- **framework/view.fw.php** : contient la classe `View` qui facilite le chargement d'une vue. Concrétement, elle ne fait que **charger un fichier** du répertoire `view` puis arrete le code avec `exit(0)`.

2.1 L'application en exemple

On désire réaliser une application simple qui demande le nom d'un utilisateur :



Presentation

Bonjour, je ne vous connais pas encore. Pouvez-vous m'indiquer votre nom ?

Donnez votre nom ici :

Puis demande de choisir une langue :

Choix de la langue

Bonjour Marcel, vous devez choisir la langue :

Dans quelle langue voulez vous être salué ? Anglais

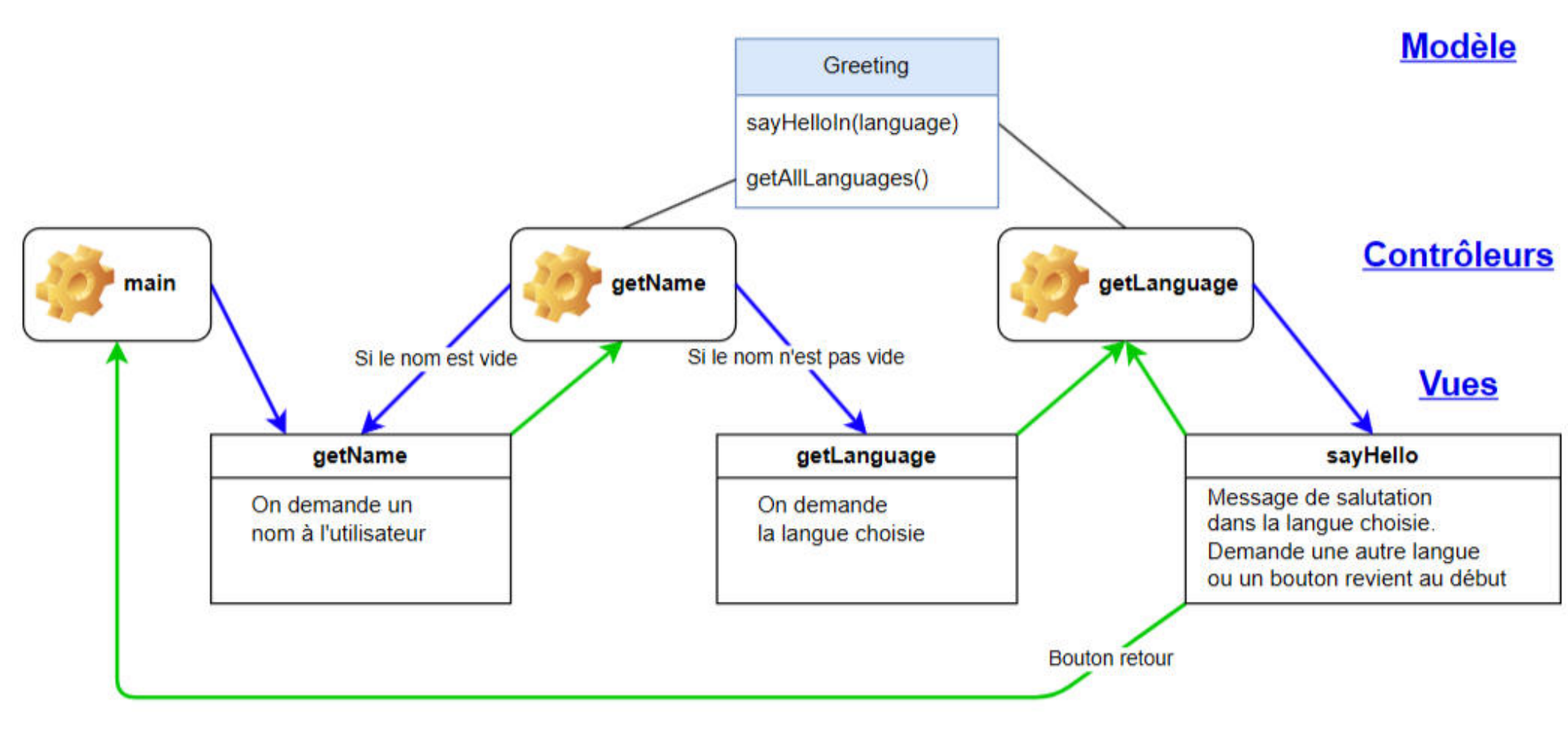
Ensuite l'application fait une salutation (greeting) dans la langue choisie :

Salutation en Anglais

Hello Marcel

Recommencer dans une autre langue ? Anglais

L'application doit être écrite avec une structure MVC. Le contrôleur est du code PHP qui réalise les actions et sélectionne la vue à afficher. Le modèle est constitué d'une seule classe `Greeting` qui sait écrire une salutation dans quelques langues (`sayHelloIn()`).



Le schéma ci-dessus indique le modèle en haut, les vues en bas et les contrôleurs au milieu. Les flèches bleues indiquent le chargement d'une vue par un contrôleur. Les flèches vertes indiquent l'appel d'un contrôleur depuis une vue. Le contrôleur `getName` accède au modèle pour avoir la liste des langues. Le contrôleur `getLanguage` accède au modèle pour l'affichage dans la bonne langue. L'application démarre avec le contrôleur `main` qui affiche la vue `getName` : elle demande un nom à l'utilisateur. Si le nom n'est pas donné (chaîne vide), alors le contrôleur recharge la même vue. Sinon, il passe à la vue qui demande la langue choisie (`getLanguage`). Une fois la langue choisie, le contrôleur `getLanguage` interroge le modèle et affiche un message de salut dans la langue choisie. On peut faire afficher un autre message dans une autre langue ou bien revenir au début par un bouton.

2.2 Le modèle

Il est composé d'une seule classe qui encapsule la constante `LANGUAGES` : elle contient pour chaque langue le texte de salutation à utiliser. A la création d'un objet `Greeting` il faut indiquer le nom de la personne à saluer. La méthode `sayHelloIn(string $language) : string` construit le texte de salutation dans la langue indiquée pour la personne données par l'objet. La **méthode de classe** `getAllLanguages()` retourne la liste de toutes les langues connues de cette classe. Cette méthode est utilisée dans les vues pour construire la liste des options à présenter dans le formulaire.

```php
<?php

// Pour dire une salutation dans quelques langues
class Greeting {
  private string $name;

  // Liste de languages
  const LANGUAGES = array (
    'Anglais' => 'Hello',
    'Allemand' => 'Hallo',
    'Arabe' => 'Salam',
    'Basque' => 'Kaixo',
    'Bulgare' => 'Zdravei',
    'Danois' => 'Hej',
    'Espagnol' => 'Hola',
    'Finnois' => 'Hei',
    'Français' => 'Salut',
    'Grec' => 'Yasou',
    'Hindi' => 'Namaste',
    'Indonésien' => 'Selamat',
    'Italien' => 'Ciao',
    'Japonais' => 'Ossu',
    'Mandarin' => 'Ní hǎo',
    'Russe' =>  'Privyèt',
    'Serbe' => 'Zdravo',
    'Ukrainien' => 'Pryvit',
    'Zoulou' => 'Sawubona'
  );

  function __construct(string $name) {
    $this->name = $name;
  }

  // Dit bonjour dans une langue
  public function sayHelloIn(string $language) : string {
    if (array_key_exists($language,self::LANGUAGES)) {
      return self::LANGUAGES[$language].' '.$this->name;
    } else {
      return '(?) '.$this->name;
    }
  }

  // Retourne la liste des langues possibles
  public static function getAllLanguages() : array {
    return array_keys(self::LANGUAGES);
  }
}

?>
```

2.3 Le routeur

Le routeur du MVC est chargé d'analyser l'URL pour savoir quel contrôleur activer. Nous choisissons le fichier `index.php` pour réaliser le routeur. Ainsi, ce fichier sera **le seul est unique** point d'entrée de l'application. Cela signifie que le chemin vers `index.php` devient **la seule URL possible**.

Nous choisissons le nom `ctrl` dans la queryString pour designer le contrôleur à activer. En l'absence de ce nom, un contrôleur par défaut est choisi. On remarque également qu'il ne faut indiquer que le nom du contrôleur : le chemin `'controler/'` et le suffixe `'.ctrl.php'` sont ajoutés par le routeur.

```php
<?php
// Calcul du routage : récupération de la route, le controleur à activer
// Par défaut on lance le contrôleur 'main'
$ctrl = $_GET['ctrl'] ?? 'main';

// Sécurité : liste des contrôleurs possibles
// Cette liste permet d'être sûr de ne pas charger un fichier inconnu
const CTRLS = array('main','getName','getLanguage');

// Sécurité : vérification que la route est correcte
if (! in_array($ctrl,CTRLS)) {
  // Ouvre une page d'erreur
  $error = 'Mauvais controleur : "'.$ctrl.'" query string "'.$_SERVER['QUERY_STRING'].'"';
  require_once('view/error.view.php');
  exit(0);
}
// Calcule le chemin vers le fichier du contrôleur
$path = 'controler/'.$ctrl.'.ctrl.php';
// Charge le contrôleur
require_once($path);
?>
```

Comme le routeur charge du code en fonction du contenu d'une URL, par sécurité, nous avons choisi de limiter la liste des fichiers que l'on peut charger. Un contrôleur non autorisé provoque un arret de l'application sur un message d'erreur.

2.4 Chargeur de vues

Pour faciliter le chargement d'une vue et le passage des paramètres à la vue, la classe `View` propose les methodes suivantes :

- **assign('nomVariable',valeur)** : permet d'ajouter une variable dans la vue et de lui donner une valeur.
- **display('nomDeLaVue')** : charge le fichier de la vue. Il faut donner le nom de la vue. le chemin `'view/'` et le suffixe `'.view.php'` sont par cette méthode. Après l'exécution de cette méthode **le code PHP se termine**/

```php
<?php

// Classe minimaliste pour normaliser l'usage d'une vue
// Cette classe est inspiré du moteur et compilateur de template Smarty
class View {
  // Paramètres de la vue, dans un tableau associatif
  private array $param;
  // Chemin vers le fichier de la vue
  private string $filePath;

  // Constructeur d'une vue
  function __construct() {
    // Initialise un tableau vide de paramètres
    $this->param = array();
  }

  // Ajoute un paramètre à la vue
  // Il n'y a aucune contrainte de type pour la valeur
  // Cela peut être par exemple un objet du modèle.
  function assign(string $varName,$value) {
    $this->param[$varName] = $value;
  }

  // Affiche la vue : on indique uniquement son nom
  function display(string $view) {

    // Ajoute le chemin relatif vers le fichier de la vue
    // ATTENTION: pour éviter de créer une variable locale $filePath qui risque
    // de se collisionner avec les variables de la vue, utilise un attribut de l'objet
    $this->filePath = "view/".$view.'.view.php';

    // Tous les paramètres de $this-param sont dupliqués en des variables
    // locales à la fonction display. Cela simplifie l'expression des
    // valeurs de la vue. Il faut simplement utiliser <?= $variable

    // Parcourt touts les paramètres de la vue
    foreach ($this->param as $key => $value) {
      // La notation $$ désigne une variable dont le nom est dans une autre variable
      // Cela crée une variable locale dont le nom est dans la variable $key
      $$key = $value;
    }

    // Inclusion de la vue
    // Comme cette inclusion est dans la portée de la méthode display alors
    // seules les variables locales à display sont visibles.
    require($this->filePath);
    // Apres cela le PHP est terminé, plus rien ne s'exécute
    exit(0);
  }
}
?>
```

2.5 Coder une vue

Une vue est principalement du HTML avec du placement de valeurs en PHP. La vue ci-dessous place les valeurs `$language`, `$salutation`, `$name` et le contenu du tableau `$languages`.

Dans un formulaire, l'appel au contrôleur doit se faire sur l'attribut `action` de `form`. Ce doit **toujours être l'URL `index.php`**. La **route**, c'est à dire le nom du contrôleur doit être indiqué dans le nom `ctrl` de la query string. Comme c'est le navigateur qui construit la query string à partir des inputs, il faut indiquer le chemin comme un input caché (hidden):

```
<input type="hidden" name="ctrl" value="NOM_CONTROLEUR"/>
```

On indique l'action qui est toujours la même :

```
<form action="index.php">
```

Dans l'URL de l'attribut `action`, est normalement composée des champs suivants :

```
<form action="http://site.web:80/chemin/vers/le/logiciel/index.php">
```

- **Le protocole** : si absent c'est `http` par défaut.
- **Le nom du site** : si absent c'est le même site que celui de la page reçue et affichée par le navigateur.
- **Le port** : si absent c'est celui de la page.
- **Le chemin vers la ressource** : si absent c'est le même chemin que la page.
- **La ressource** : si absente, c'est le même nom de ressource que la page.

En fin de compte, toutes les URL de l'application qui activent du code PHP dans l'attribut `action` sont une seule et même URL : le chemin vers le fichier `index.php`. Ce n'est donc même plus la peine de l'indiquer !

```html
<html>
<head>
  <meta charset="UTF-8">
</head>
<body>
  <h1>Salutation en <?= $language ?></h1>
  <p> <?= $salutation ?> </p>
  <form>
    <input type="hidden" name="ctrl" value="getLanguage"/>
    <input type="hidden" name="name" value="<?=$name?>"/>

    <p>Recommencer dans une autre langue ?
      <select name="language">
        <?php foreach($languages as $language) : ?>
          <option value="<?= $language ?>"><?= $language ?></option>
          <?php endforeach; ?>
      </select>
    </p>
    <button type="submit">Saluer</button>
  </form>
  <a href="index.php"><button type="button">Retour au début</button></a>
</body>
</html>
```

2.6 Coder un contrôleur

Le contrôleur doit include le code du framework et du modèle. Il doit récupérer les informations de la query string, lancer les calculs avec le modèle, puis mettre en place une vue en lui passant les valeurs à afficher. Il ne faut pas traiter le nom `ctrl` car il est déjà traité par le routeur.

```php
<?php
require_once('framework/view.fw.php');
require_once('model/greeting.class.php');

//////////////////////////////////////////////////////////////////////////////
// PARTIE RECUPERATION DES DONNEES
//////////////////////////////////////////////////////////////////////////////

// Récupération des informations de la query string
$name = $_GET['name'] ?? '';

//////////////////////////////////////////////////////////////////////////////
// PARTIE SELECTION DE LA VUE
//////////////////////////////////////////////////////////////////////////////

// Création de la vue
$view = new View();

// Si le nom est vide on retourne à la page de lecture du nom
if ($name == '') {
  $view->display('getName');
} else {
  $view->assign('name', $name);
  // Recupère la liste des langues à partir du modèle
  $view->assign('languages',Greeting::getAllLanguages());
  // Vue pour faire choisir la langue
  $view->display('getLanguage');
}
```

À faire :

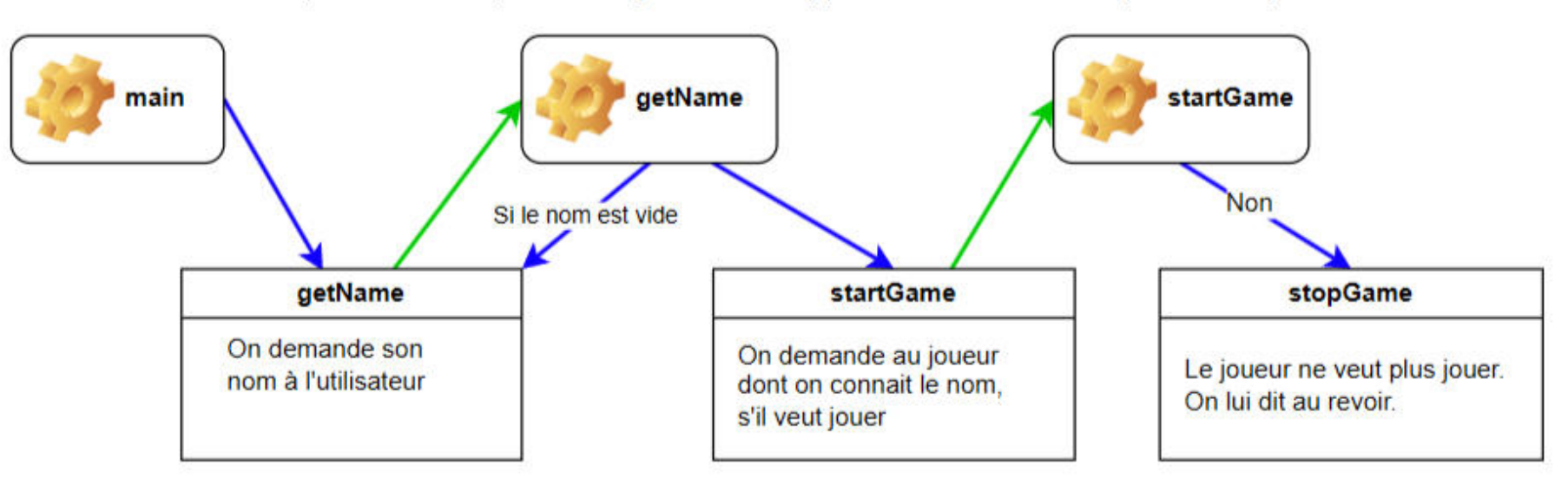
- Télécharger data/hello_MVC.zip ,
- Faire fonctionner cette application et lire attentivement le code source.

3. Jeu du nombre en MVC et avec une session

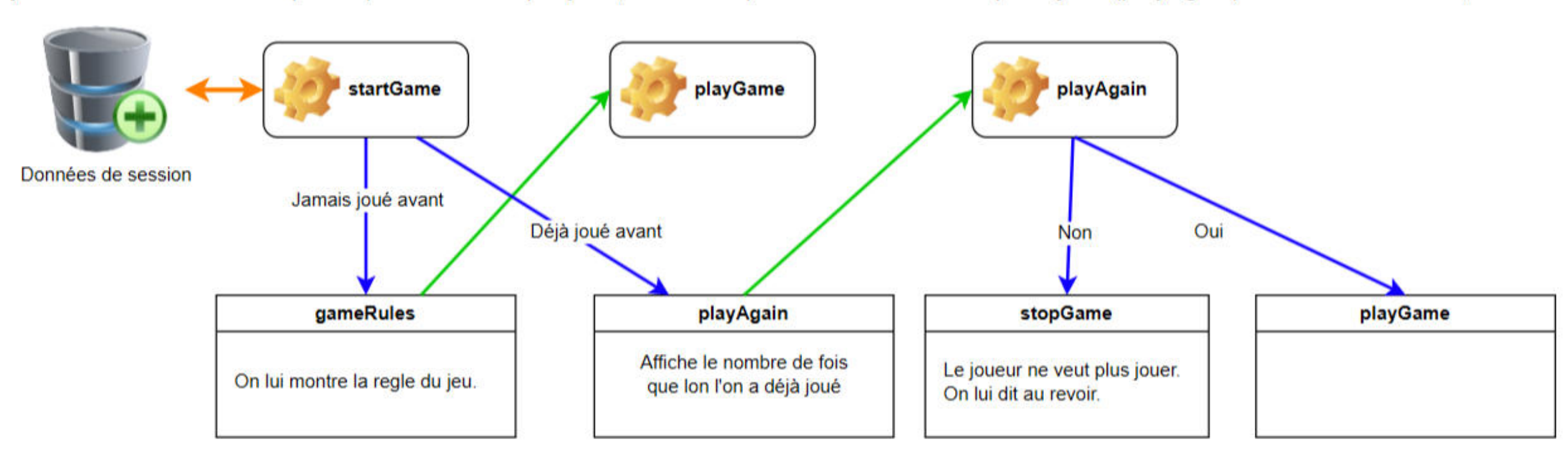
Dans cette partie, vous allez examiner un exemple de programme PHP structuré selon le motif MVC. Ce programme est un jeu de recherche, par la machine, d'un nombre caché par l'utilisateur. La recherche se fait de manière dichotomique.

Au début de la partie (main), l'application demande son nom à l'utilisateur (getName). Tant qu'il donne un nom vide, l'application reste sur cette demande. Dès que l'on connait son nom, on lui demande s'il veut jouer (startGame). S'il ne veut pas jouer, on affiche en écran d'au revoir (stopGame).

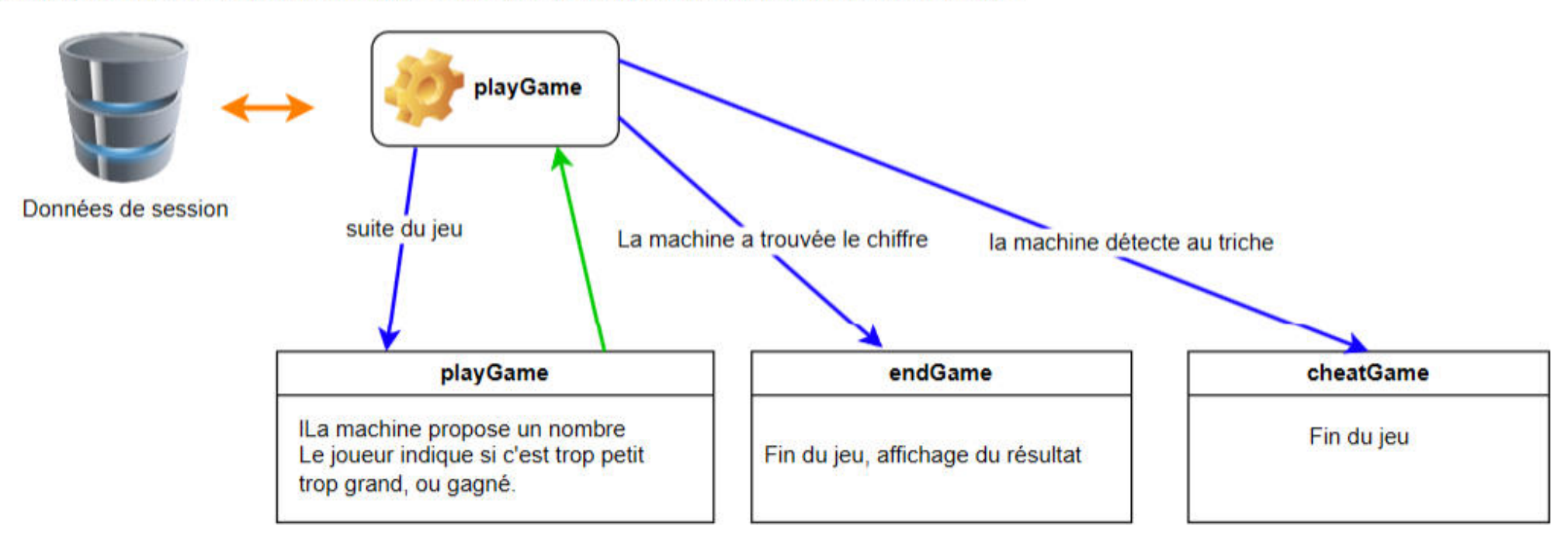
Le diagramme ci-dessous indique ce fonctionnement et montre l'alternance entre les affichages HTML (en bas) et les contrôleurs PHP (en haut). Les affichage HTML sont les `vues`. Ce sont les contrôleurs en PHP (en haut) qui chargent et choisissent une vue (flèche bleue). La vue peut faire appel à un contrôleur, par exemple si elle contient un formulaire (form). C'est la flèche verte.



Si l'utilisateur accepte de jouer, une instance de la classe `Game` est créée et stockée dans la session. Si une instance existait déjà, on la récupère et on ré-initialise le jeu. Si c'est la première fois qu'il joue, on lui donne la règle du jeu (gameRules), et le jeu est lancé (playGame). Comme le joueur n'a pas de choix dans sa réponse, il est inutile de créer un contôleur spécifique pour cette vue : on se rend directement au jeu lui même. Si ce n'est pas la première fois qu'il joue, on lui indique le nombre de fois qu'il a joué (playAgain). Dans les deux cas, on lance le jeu (playGame)



Finalement, le jeu se déroule avec deux fins possibles : soit la machine a trouvé le nombre, soit l'utilisateur a triché.



Travail à réaliser :

- Téléchargez le fichier data/jeux_recherche_nombre_MVC.zip
- Installez les fichiers de cette archive dans votre répertoire WEB, puis testez ce programme.
- Ouvrez un shell, allez dans le répertoire `model`, puis lancez le test de la classe `Jeux`. Vérifiez que le résultat de ce test est correct.
- Examinez le fichier `getName.ctrl.php` du répertoire `controler` :
  - À quoi sert la variable `nom` ?
  - Comment est chargée la vue quand le `nom` est inconnu ?
  - Comment les paramètres à afficher dans une vue sont-ils passés à cette vue ?
- Examinez le fichier `view.fw.php` dans le répertoire `framework` :
  - Où sont stockés les paramètres de la vue ?
  - Quelle instruction permet de charger le fichier de la vue proprement dite ?
  - Quelle conséquence sur les variables a le fait que cette instruction se trouve à l'interieur du code de la méthode `display` ?
  - Quelle est la conséquence de l'instruction `exit(0)` à la fin de `display` ?
  - À quoi sert la notation `$$` de la variable `key` et quelle est la différence avec `$key` ?
- Examinez le fichier `startGame.view.php` dans le répertoire `view`:
  - Comment est affiché le paramètre de la vue ?
  - Pourquoi il n'y a pas d'attribut `action` dans dans balise `form` ?
  - Une vue ne peut déclencher **que des contrôleurs**. Quel contrôleur va déclencher le formulaire ?
  - Dans le formulaire, à quoi sert l'entrée `input` avec le type `"hidden"` ?
  - Pourquoi y a-t-il un chemin relatif dans l'attribut `href` du fichier de style (balise `link`) ? À quoi est 'relatif' ce chemin ?
  - Pourquoi n'indique-t-on aucun chemin pour le fichier de la balise `action` du formulaire ?
- Examinez le fichier `playAgain.view.php` :
  - Commee `form` ?
  - Commee `form` ?
  - En général, quelle est la différence dans un formulaire entre une balise `input` de type `submit` et une balise `button` de type `submit` ?
  - À quoi sert l'attribut `formaction` du deuxième bouton ?
- Examinez le fichier `playGame.ctrl.php` :
  - Comment sont conservées les informations sur le jeu en cours ?
  - Que contient le tableau `$_SESSION` ?
  - Comment la machine sait-elle que le joueur à triché ? (examiner aussi `game.class.php`)
  - Pourquoi, à la fin du contrôleur, dans les tests de choix de la vue, il n'y a pas d'imbrication des cas (i.e. pas de `if` imbriqués) ? (examiner `view.class.php` pour comprendre)
  - Dans quel cas peut-il y avoir une erreur ? Et comment est-elle traitée ?
  - Est-ce que l'instruction `session_write_close()` est indispensable ? pourquoi ?
- Examinez `startGame.ctrl.php` :
  - Quels sont les deux cas d'erreur ?
  - Dans quel cas la méthode `restart` est-elle appelée et à quoi sert-elle ?
  - À quoi sert la fonction `session_destroy()` ? Quelle conséquence a-t-elle sur le comportement du jeu ?
- À quoi sert le fichier `jeu.test.php` ?
- Modifier le jeu pour intégrer le nom du joueur dans la session. Il ne sera alors plus la peine de le passer de page en page dans les formulaires.

Sujet de TP Programmation WEB (PHP).