Projet Serval (PHP – POO)

Application de vue à la première personne

Html – CSS – PHP - MySQL

**Partie 1**

La **vue à la première personne** (souvent appelé **vision subjective**) désigne une façon d'apercevoir l'environnement d’un décor (pièces, autres personnages, etc.) comme si l’utilisateur incarnait le personnage qu'il contrôle en voyant la scène à travers les yeux de son avatar.



L’application permet de créer un tel avatar qui parcourt un musée, une habitation, un donjon, ou un lieu comme dans Street View. Elle repose sur deux classes FirstPersonView et FirstPersonText héritant d’une classe de base BaseClass.

La carte (map) dans laquelle l’avatar se déplace est constituée de toutes les coordonnées des positions disponibles avec 4 points de vue pour chacun d’eux (Est, Nord, Ouest, Sud) de sorte que chaque image représente ce que voit l’avatar quand il est dirigé vers un point de vue.

 \*

 \*   5             ^

 \*   4           90

 \*   3     c < 180   0 >

 \*   2  a  b       270

 \*   1             v

 \* X 0  1  2  3  4  5  6 ...

 \*   Y

 \*

1. Si votre avatar se trouve en coordonnées [1, 2] avec un angle de vue 0 (vers l’Est) et qu’il se déplace vers l’avant, l’algorithme va le placer en coordonnées [2, 2] avec un angle de vue 0.
2. Si votre avatar se trouve en coordonnées [2, 2] avec un angle de vue 0 et qu’il tourne à droite, l’algorithme va le placer en coordonnées [2, 2] avec un angle 270 (vers le Sud)
3. Si votre avatar se trouve en coordonnées [2, 2] avec un angle 270 et qu’il se déplace en arrière, l’algorithme va le placer en coordonnées [2, 3] avec un angle 270.

Et ainsi de suite.

Le layout



L’écran se compose de trois parties :

* La vue à la première personne
* La zone de gestion des mouvements comportant
  + Un bloc de 6 boutons permettant à l’avatar de se déplacer vers l’avant, sur le côté droit, sur le côté gauche, en arrière, de tourner à droite ou à gauche. Lorsqu’une coordonnée n’est pas disponible, le bouton menant vers cette coordonnée est désactivé.
  + Une boussole indiquant la direction de la vue. Elle se met en position à chaque changement de direction.
  + Une zone de texte affichant les informations correspondant à l’emplacement sur la carte et éventuellement l’angle de vue (le texte peut être différent selon que l’avatar regarde vers l’Est ou le Nord, etc.)

La carte, les chemins vers les fichiers images et les textes sont stockés dans une base de données : « fpview »

Connexion à la base de données

La base «fpview » contient 3 tables :

|  |  |
| --- | --- |
| images | contient les chemins vers les fichiers images correspondants aux différentes vues (coordonnées sur la carte et angles de vue) |
| map | contient toutes les coordonnées accessibles par l’avatar |
| text | contient tous les textes correspondants aux différentes coordonnées et angles de vue |

La table map contient 4 colonnes :

|  |  |  |
| --- | --- | --- |
| **Nom** | **Type** | **Description** |
| id | int | Clé primaire auto-incrémentée |
| coordx | int | Coordonnée X d’un point accessible |
| coordY | int | Coordonnée Y d’un point accessible |
| direction | int | Angle de vue (0, 90, 180 ou 270) |

La table images contient 3 colonnes :

|  |  |  |
| --- | --- | --- |
| **Nom** | **Type** | **Description** |
| id | int | Clé primaire auto-incrémentée |
| map\_id | int | Clé étrangère liée à la table map |
| path | varchar(250) | Le chemin vers l’image pour cette map\_id |

Tips : fixer une nomenclature pour les noms de fichier, par exemple : xy-angle.jpg où x, y et angle sont respectivement la coordonnée X, la coordonnée Y et la direction de vue.

La table text contient 3 colonnes :

|  |  |  |
| --- | --- | --- |
| Nom | Type | Description |
| id | int | Clé primaire auto-incrémentée |
| map\_id | int | Clé secondaire liée à la table map |
| text | int | Le texte à afficher pour cette map\_id |

La classe DataBase

La classe DataBase permet la créer la connexion à la base de données. Elle hérite de la classe native PDO.

Elle comprend 4 constantes de classe respectivement associées à l’hôte (localhost), l’utilisateur (root), le mot de passe (root) et le nom de la base (fpview).

La classe possède un constructeur public faisant appel au constructeur de la classe parente (PDO)

La classe BaseClass

La classe mère BaseClass permet de créer un avatar « vue à la première personne » et de gérer les mouvements et l’affichage des images pour chaque point de la map. Elle possède au minimum les propriétés suivantes (de visibilité protected) :

|  |  |
| --- | --- |
| $\_currentX | (int) Coordonnée X sur la carte |
| $\_currentY | (int) Coordonnée Y sur la carte |
| $\_currentAngle | (int) Angle de vue |
| $\_dbh | (object) La connexion à la base de données |

Cette classe possède au moins les méthodes publiques suivantes :

* Un constructeur public ayant pour fonction d’initialiser la connexion à la base de données.
* Les setters et les getters publics associés aux propriétés $\_currentX, $\_currentY, $\_currentAngle.
* Six méthodes publiques permettant de tester si un déplacement dans les quatre directions ou le changement d’angle de vue est possible :

|  |  |  |
| --- | --- | --- |
| **Nom** | **Retourne** | **Description** |
| checkForward | bool | Vérifie la possibilité de déplacement vers l’avant |
| checkBack | bool | Vérifie la possibilité de déplacement vers l’arrière |
| checkRight | bool | Vérifie la possibilité de déplacement vers la droite |
| checkLeft | bool | Vérifie la possibilité de déplacement vers la gauche |
| checkTurnRight | bool | Vérifie la possibilité de tourner à droite |
| checkTurnLeft | bool | Vérifie la possibilité de tourner à gauche |

**Un déplacement n’est possible que s’il existe une entrée en base de données correspondant au résultat de ce déplacement.**

Tips : Il sera utile d’implémenter une méthode privée \_checkMove prenant trois paramètres (coordonnées x et y cibles et l’angle de vue) afin d’interroger la base de données et de vérifier que le mouvement vers les coordonnées cibles est bien possible.

|  |  |  |
| --- | --- | --- |
| **Nom** | **Retourne** | **Description** |
| \_checkMove | bool | Vérifie la possibilité de déplacement vers une position cible |

* Six méthodes publiques permettant d’effectuer un déplacement dans les quatre directions et de changer l’angle de vue :

|  |  |
| --- | --- |
| **Nom** | **Description** |
| goForward | Effectue le déplacement vers l’avant |
| goBack | Effectue le déplacement vers l’arrière |
| goRight | Effectue le déplacement vers la droite |
| goLeft | Effectue le déplacement vers la gauche |
| turnRight | Tourne sur la droite |
| turnLeft | Tourne sur la gauche |

La classe FirstPersonView

La classe FirstPersonView spécialise la classe BaseClass en lui ajoutant les fonctionnalités de gestion des images et de la boussole. Elle hérite donc de BaseClass.

Elle possède :

* une constante de classe indiquant le répertoire dans lequel sont placées les images.
* à minima la propriété protected :

|  |  |
| --- | --- |
| $\_mapId | (int) l’identifiant de la position courante sur la carte |

* deux méthodes publiques :

|  |  |  |
| --- | --- | --- |
| **Nom** | **Retourne** | **Description** |
| getView | string | Renvoie le chemin vers le fichier image à afficher |
| getAnimCompass | string | Renvoie la direction vers laquelle pointe la boussole. Il pourra s’agir de la classe CSS indiquant l’image de la boussole à afficher. |

La classe FirstPersonText

La classe FirstPersonText spécialise la classe BaseClass en lui ajoutant les fonctionnalités de gestion des textes à afficher. Elle hérite donc de BaseClass.

Elle possède

* Une méthode publique :

|  |  |  |
| --- | --- | --- |
| **Nom** | **Retourne** | **Description** |
| getText | string | Renvoie le texte à afficher |

Méthodologie

1) Réaliser le MCD sur les 3 tables : map (points accessibles de la carte), text (les textes à afficher), images (les images à afficher).

2) Créer une base de données « fpview » et les tables selon le MCD

3) Dans un fichier database.php, développer une classe Database qui hérite de la classe PDO native de PHP. Les propriétés de la classe sont les paramètres de connexion PDO à la base. La classe ne possède qu’une seule méthode : son constructeur dans lequel la connexion est réalisée.

4) Créer dans le fichier index.php une fonction d’autoload.

5) Toujours dans le fichier index.php, coder le layout (partie html : first person view, le bloc de bouton et l’affichage du texte et de la boussole. Le bloc de boutons peut être constitué d’une table insérée dans un formulaire. Au clic sur l’un des boutons de déplacement / rotation le formulaire est soumis.

5) Modifier le fichier index.php (partie php) de façon à traiter le déplacement transmis depuis le formulaire précédent. Le fichier index.php contient une instanciation des classes FirstPersonView et FirstPersonText permettant l’affichage du décor et des textes en fonction de la position sur la map.

Prévoir la position initiale de l’avatar en début de parcours alors qu’aucun formulaire n’est encore soumis.

6) Dans un fichier FirstPersonView.class.php développer la classe firstPersonView.

7) Dans un fichier FirstPersonText.class.php développer la classe firstPersonText.

8) Selon le principe DRY (Don’t Repeat Yourself) factoriser le code : toutes les instructions répétées dans le code peuvent être regroupées dans une fonction.

**Partie 2**

Dans cette partie nous allons ajouter des fonctionnalités permettant d’interagir avec l’environnement.

De base on définit 2 actions :

* Prendre (take) : le personnage ramasse un item présent sur la map et le met dans son inventaire.
* Utiliser (use) : le personnage utilise l’item (qui doit donc être requis) à la position courante

Lorsqu’une action est possible, le bouton central X est actif :

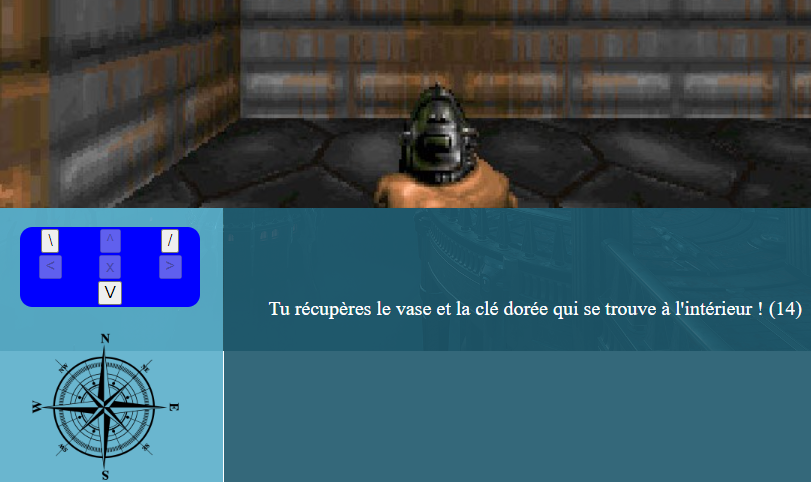


Une action est possible si

* Un objet est disponible sur la map
* L’avatar possède un objet requis dans son inventaire
* Le statut de la map (coordonnées + angle de vue) est égal au statut de l’action.

Par exemple :

* Une clé est disponible en map\_id = 14, aucun objet n’est requis, le statut de la map et de l’action valent 0 ; Le bouton action (X) est activé.
* L’utilisateur clique sur le bouton action
* L’inventaire de l’avatar est mis à jour
* Le statuts de la map passe à 1
* Une nouvelle image correspondant à l’id map (14) et dont le statut est égal à 1 dans la table images est affichée. De même un nouveau texte dont le statut est égal à 1 dans la table text est affiché.



A moins qu’il existe en base une action pour l’id map 14 et dont le statut est égal à 1 (statut courant de la map depuis que la précédente action a été réalisée), l’action suivante sera possible, sinon le bouton d’action est forcément désactivé puisque plus aucune action n’est disponible à cet endroit.

Modification du MCD

On ajoute deux tables au précédent modèle.

|  |  |
| --- | --- |
| actions | contient les actions correspondantes aux coordonnées sur la carte et angles de vue |
| items | contient tous les items présents sur la carte avec lesquels l’avatar peut interagir |

La table actions contient 6 colonnes

|  |  |  |
| --- | --- | --- |
| **Nom** | **Type** | **Description** |
| id | int | Clé primaire auto-incrémentée |
| map\_id | int | Clé secondaire liée à la table map |
| action | varchar(50) | Le type d’action ‘take’ ou ‘use’ |
| item\_id | int | Clé secondaire liée à la table items |
| requis | int | L’identifiant de l’item requis (présent dans l’inventaire). 0 si rien est requis |
| status | int | Le statut de l’action, 0 par défaut |

La table items contient 2 colonnes :

|  |  |  |
| --- | --- | --- |
| **Nom** | **Type** | **Description** |
| id | int | Clé primaire auto-incrémentée |
| description | varchar(50) | La description de l’item |

Il faut modifier

* la table ‘images’ en ajoutant une colonne ‘status\_action’, valeur par défaut 0
* la table ‘map’ en ajoutant une colonne ‘status\_action’, valeur par défaut 0
* la table ‘text’ en ajoutant une colonne ‘status\_action’, valeur par défaut 0.

La classe FirstPersonAction

La classe FirstPersonAction spécialise la classe BaseClass en lui ajoutant les fonctionnalités de gestion des actions avec le décor. Elle hérite donc de BaseClass et possède au moins 2 méthodes publiques.

|  |  |  |
| --- | --- | --- |
| **Nom** | **Retourne** | **Description** |
| checkAction | bool | Détermine en fonction de la position sur la map si les conditions sont requises pour réaliser une action. Si l’action n’est pas possible, le bouton d’action est désactivé. |
| doAction |  | Réalise l’action et modifie la base de données en conséquence (inventaire, items, etc.) |

Méthodologie

1) Modifier le MCD sur les 5 tables décrites ci-dessus.

2) Modifier la base de données « fpview » et les tables selon le MCD

3) Dans un fichier FirstPersonAction.class.php développer la classe firstPersonAction.

4) Selon le principe DRY (Don’t Repeat Yourself) factoriser le code : toutes les instructions répétées dans le code peuvent être regroupées dans une fonction.

L’inventaire de l’avatar pourra être stocké dans un tableau $\_SESSION

Prévoir qu’en début de parcours, l’inventaire doit être vide et tous les statuts des points de la map sont à 0.

La table action contient 6 colonnes :

|  |  |  |
| --- | --- | --- |
| **Nom** | **Type** | **Description** |
| id | int | Clé primaire auto-incrémentée |
| map\_id | int | Clé étrangère de la table map |
| action | varchar(50) | Le nom de l’action |
| item\_id | int | Clé étrangère de la table item |
| requis\_id | int | Clé étrangère de la table item si l’item est requis pour réaliser l’action |
| image | varchar(250) | Chemin vers le fichier image à afficher après réalisation de l’action |