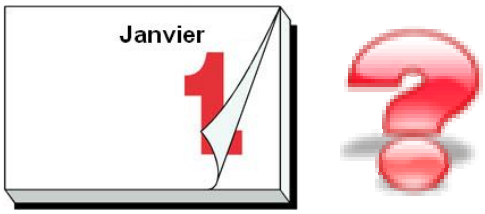
UFR de Sciences
Département d'informatique

Licence d'informatique
Année universitaire 2002-2003

Projet EIAO

EducaTemps



« Se repérer, reconnaître, orthographier »

Julien VAN DEN BOSSCHE
Benoît MOULIN

Sous la direction de Françoise Lambert

Sommaire

1. Introduction ……...…….page 2
1.1. A qui sert l'EIAO ……..…….page 2
1.2. Pourquoi ce logiciel ? ……page 2

2. Un logiciel pour se repérer, reconnaître, apprendre .page 3
2.1. Les normes à respecter …….page 3
2.2. Un logiciel ouvert à d'autres exercices ……page 3
2.3. Description du modèle ……..page 3
2.3.1. Les acteurs ……..page 3
2.3.2. Les informations échangées ……..page 4
2.4. Une session type : différents scénarios ……....page 5
2.4.1. Pour l'élève ……..…page 5
2.4.2. Pour l'éducateur ……..…..page 10

3. Modélisation objet …….……page 14
3.1. Diagramme de classes …….....page 15
3.2. Descriptions des différentes classes ……..page 16

4. Modèle dynamique, trace des événements...…….....page 30

5. Gestion de la base de données……page 34

6. Conclusion……page 36

1. Introduction :

1.1. A qui sert l'EIAO ?

Certains enfants en difficultés n'ont pas la possibilité de suivre une scolarité normale. Ils suivent alors une éducation spécialisée. Ils sont pris en charge par des éducateurs et sont accueillis dans des établissements parmi lesquels on trouve les instituts médico-pédagogiques (IMP). Pour aider ces enfants et adolescents, les IMP utilisent plusieurs méthodes de travail, notamment l'enseignement intelligemment assisté par ordinateur (EIAO). C'est le cas de l'IMP de St Sever dans le Calvados, pour lequel nous avons créé un logiciel que nous allons vous présenter à travers ce rapport.

1.2. Pourquoi ce logiciel ?

Durant notre année de licence d'informatique, nous devons réaliser un projet. Nous avons choisi le thème de l'EIAO, car c'est très intéressant et motivant de réaliser un logiciel qui sera utilisé pour apprendre à des enfants certaines choses essentielles de la vie de tous les jours, et qui nous semblent, à nous, tellement simples. De plus, ce sujet nous laisse une certaine liberté dans nos choix. Chaque binôme qui aura travaillé sur ce thème, aura un logiciel différent, ce qui est idéal pour l'IMP.

Notre logiciel a pour thème principal le temps. En rencontrant les responsables de l'institut, il nous ont expliqué que les enfants ne connaissent pas le référentiel temps et donc ils ne peuvent pas se situer dans une semaine ou bien dans une année. Ils n'arrivent pas à lier le temps aux événements présents et ne peuvent pas quantifier une durée. Le but de notre logiciel n'est pas d'apprendre l'heure, mais de repérer les jours dans une semaine, les mois dans une année, ainsi que les saisons. Par exemple, s'ils regardent une photographie montrant des feuilles mortes, ils ne savent pas de quelle saison il s'agit.

Plan

Dans un premier temps, nous présenterons le logiciel en analysant les problèmes qui se sont posés à nous et en montrant comment nous les avons résolus. Dans cette première partie, nous expliquerons le fonctionnement du logiciel en décrivant une session type, d'abord du point de vue de l'élève, puis de celui de l'éducateur.

Ensuite, dans une seconde partie, nous exposerons nos choix techniques. Nous justifierons nos classes et nos objets. Nous présenterons également les outils que nous avons utilisés pour créer notre interface graphique et nous expliquerons comment est gérée la base de données.

2. Un logiciel pour se repérer, reconnaître, apprendre

2.1. Les normes à respecter

Au cours de l'année, notre responsable de projet, Françoise Lambert nous a réuni avec Madame Ozenne, responsable de l'IMP de St Sever, qui était accompagnée de l'un des ses collègues. Ces rencontres nous ont permis d'obtenir des informations concernant les consignes à respecter par rapport à l'interface. En effet, à cause des faibles notions d'orthographe des élèves, chaque page du logiciel doit être simple, visuelle et avec de gros caractères. Les énoncés doivent être concis et avec des mots qui ne doivent pas être trop difficiles à lire. Pour annoncer à l'élève si sa réponse est correcte ou pas, il ne faut pas afficher de trop longues phrases, mais nous devons utiliser des couleurs (par exemple, vert si c'est correct, rouge sinon). Etant donné le faible temps de concentration des élèves, nous devons mettre en place des moyens permettant de sortir de l'exercice, pour soit arrêter la séance, soit changer d'activité.

Nous devons permettre aussi d'enregistrer le travail en cours pour une prochaine session pour ne pas que l'élève fasse toujours les mêmes questions. Il nous a été également demandé de prévoir une évaluation pour que l'éducateur puisse se rendre compte du travail effectuer par l'élève et pour qu'il puisse analyser les échecs et les réussites de l'élève sur certains exercices. De plus, il est essentiel pour l'élève d'avoir un compte rendu de son évolution, surtout si celle-ci est positive.

2.2. Un logiciel ouvert à d'autres exercices

Le logiciel que nous présentons est composé d'exercices qui posent des questions en rapport avec le temps. Ce thème était une attente de l'IMP. Les exercices que nous avons créés peuvent aussi servir pour étudier des mots provenant de différents thèmes. Par exemple, si les éducateurs décident de faire travailler les élèves sur le sport, ils peuvent utiliser des images en rapport avec le sport pour leur apprendre à écrire des mots comme tennis, football ou planche à voile. **EDUCATEMPS** n'est donc pas un logiciel « figé » et c'est cette particularité que nous avons voulu privilégier en programmant ce projet.

2.3. Description du modèle

2.3.1. Acteurs

Ce logiciel concerne deux catégories d'acteurs, qui sont en fait les utilisateurs, c'est-à-dire les élèves et les éducateurs. Le but de l'élève est d'apprendre, à l'aide du logiciel, en répondant aux questions demandées dans les différents exercices. En ce qui concerne l'éducateur, son objectif correspond à celui du logiciel : enseigner à l'élève. Son rôle est de guider l'élève lors d'une séance de travail, de l'évaluer et aussi de créer de nouveaux exercices. Il est noter que lors d'une session un élève est toujours accompagné d'un éducateur.

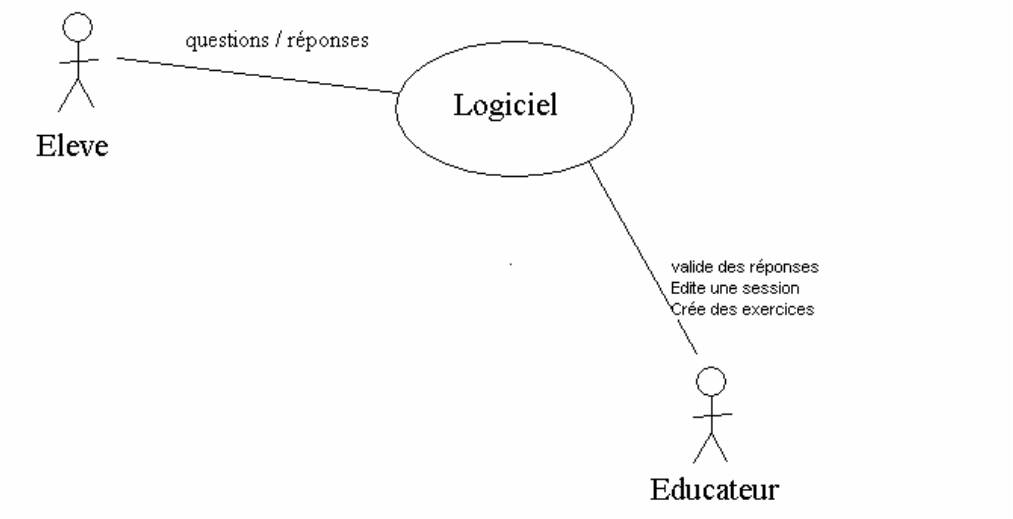
2.3.2. Informations échangées selon les cas d'utilisation

Les informations échangées entre le système et les acteurs correspondent en fait à des réceptions de messages, par les acteurs, et à l'émission d'autres messages vers le système. Dans chaque cas d'utilisation, il y a échanges de messages.
L'élève n'utilise le logiciel que d'une seule manière : lorsqu'il apprend. Il existe trois types d'échanges entre l'élève et le système. Tout d'abord, l'élève reçoit des questions, des consignes de la part du logiciel. Ensuite, il émet des réponses pour résoudre les exercices. Enfin, il reçoit un message qui lui signale si sa réponse est vraie ou fausse.
Pour l'éducateur, il y a trois cas d'utilisation possibles. Le premier consiste à évaluer l'élève pendant la séance de travail. L'évaluation est donc visuelle. L'éducateur reçoit (lit) la réponse de l'élève et émet la validation de la réponse. Il peut valider une réponse qui n'est pas correcte s'il considère que l'élève a fait un important effort de réflexion pour arriver à un résultat quasi-parfait (notamment pour ce qui concerne les exercices d'écriture). Le deuxième cas est également une évaluation mais qui s'effectue après la séance de travail. L'éducateur reçoit alors des informations par l'intermédiaire de l'édition de la session de travail de l'élève. Le troisième et dernier cas d'utilisation pour l'éducateur concerne la création d'exercices. En effet, il peut créer de nouveaux exercices pour chaque catégorie et chaque niveau.

Tableau récapitulatif :

Cas d'utilisation	Acteurs	Informations échangées
Apprentissage	Elève	Reçoit : questions / validations Emet : réponses
Evaluation visuelle	Educateur	Reçoit : réponses de l'élève Emet : validation si nécessaire
Evaluation de la séance	Educateur Elève	Reçoit : l'ensemble des résultats des exercices
Création	Educateur	Reçoit : structure de l'exercice Emet : données (images,réponses …)

Diagramme :



2.4. Session type : Les différents scénarios

Comme nous l'avons déjà dit, notre logiciel a pour but d'apprendre à repérer et à écrire les jours de la semaine ainsi que les mois et les saisons. Nous l'avons appelé **EDUCATEMPS**. Nous avons créé trois types d'exercices. Tout d'abord, les élèves peuvent travailler sur l'orthographe des jours, mois et saisons. Ensuite, nous avons créé des exercices qui utilisent des images. Les élèves doivent associer un mot (jour, mois ou saison) à une image. Enfin, le dernier type d'exercice demande aux élèves de répondre à des questions sur la semaine ou sur l'année. Notre logiciel est également destiné aux éducateurs. Ils peuvent créer ou supprimer des exercices et le logiciel leur permet d'évaluer les élèves. Nous allons vous expliquer le fonctionnement d'**EDUCATEMPS** en vous décrivant comment les acteurs (élèves et éducateurs) peuvent l'utiliser.

2.4.1. L'élève

Scénario 1 : Ouverture de la session

Pour travailler avec **EDUCATEMPS**, l'élève doit d'abord s'identifier. Lors de sa première utilisation, il doit écrire son nom et son prénom. Ceux-ci sont alors mémorisés et l'élève pourra alors retrouver son nom dans une liste.

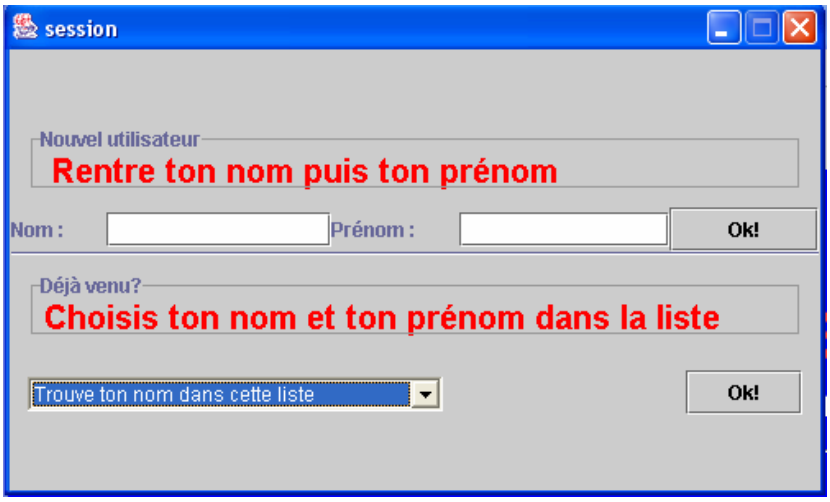


Figure 1 : Ouverture d'une session.

Scénario 2 : Choix d'un exercice

Une fois identifié, l'élève arrive sur la page d'accueil :

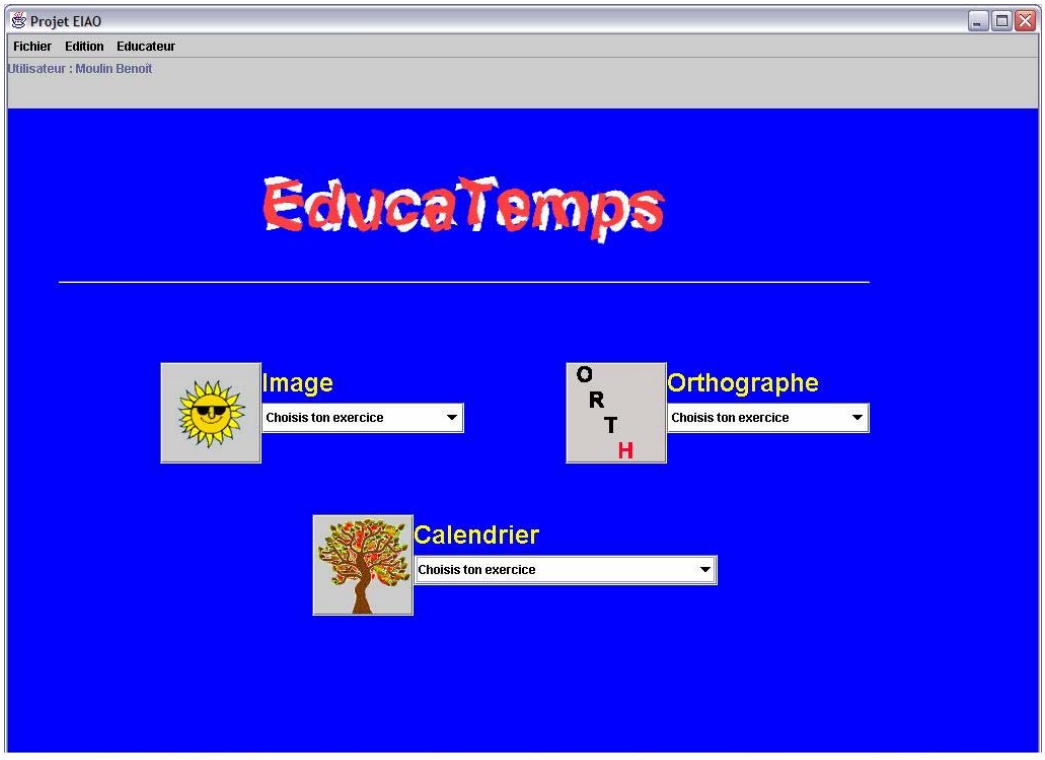


Figure 2 : Page d'accueil.

Sur cette page, l'élève avec l'accord de l'éducateur, doit d'abord choisir la catégorie (**Orthographe**, **Image** ou **Calendrier**) dans laquelle il va travailler. Ensuite, dans la liste de la catégorie, il doit choisir l'exercice. Celui-ci est défini par son titre et son niveau qui correspond à un chiffre entre parenthèses ((1) pour facile, (2) pour moyen et (3) pour difficile). Une fois ce choix établi, l'élève clique sur l'image pour lancer l'exercice.

Figure 3 : Choix de l'exercice.

Scénario 3 : L'élève travaille sur un exercice du type **Orthographe**

Le but d'un exercice de cette catégorie est de reconnaître l'orthographe correcte d'un mot (jour, mois ou saisons). L'élève doit choisir entre plusieurs propositions.

*Figure 4 : Exercice de la catégorie **Orthographe**.*

Au niveau facile, l'élève doit juste cliquer sur la réponse qu'il pense correcte. Au niveau difficile, il doit recopier sa réponse. Lorsqu'il a fourni son choix, une fenêtre de validation apparaît. Si la réponse est correcte, cette fenêtre est composée d'un personnage vert et d'une phrase de félicitation.

Figure 5 : Fenêtre de validation.

Sinon le personnage est rouge :  .

Scénario 4 : L'élève travaille sur un exercice du type **Reconnaissance d'image**

L'objectif est d'associé un mot à une image. Un exercice est composé d'une image, d'une liste de solutions possibles et d'une case où l'élève doit donner sa réponse.

*Figure 6 : Exercice de la catégorie **Image**.*

Il existe trois niveaux pour ce type d'exercice. Au premier, l'élève doit faire glisser sa réponse de la liste des propositions vers l'emplacement situé sous l'image. Il valide en cliquant sur le bouton **OK**. Au niveau moyen, l'élève doit recopier sa réponse. En mode difficile, il doit aussi écrire sa réponse mais il n'a plus de modèle. Lorsqu'il a confirmé sa réponse, la fenêtre de validation (cf. figure 5) apparaît.

Scénario 5 : L'élève travaille sur un exercice du type **Repérage**

Cette catégorie regroupe des exercices qui posent diverses questions aux élèves, comme par exemple : « *Quelle sont les mois de l'hiver ?* » ou « *Quels sont les jours du week-end ?* ». Comme pour la catégorie précédente, il y a une liste de propositions et des cases réservées aux réponses.

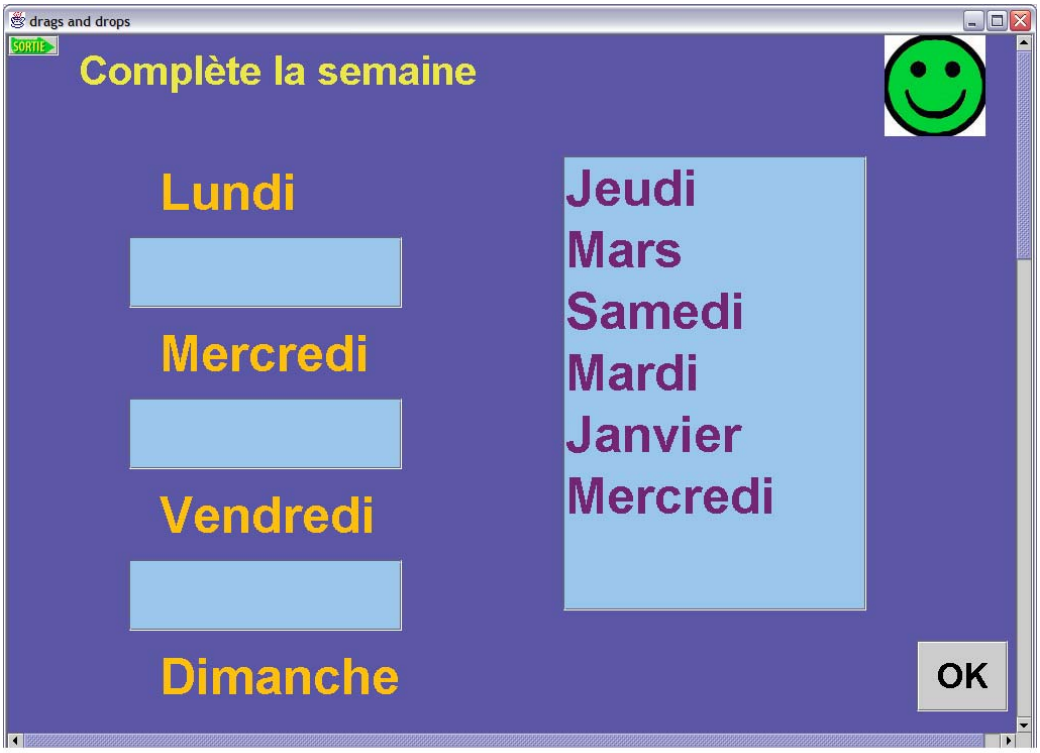


*Figure 7 : Exercice de la catégorie **Calendrier**.*

Il y a également trois niveaux : facile où il faut déplacer la réponse, moyen où il faut la recopier et difficile où il faut l'écrire sans avoir de modèle. La validation se fait toujours par le bouton **OK**, ce qui fait apparaître la fenêtre de la figure 5.

Scénario 6 : L'élève quitte un exercice

Si l'élève décide, avec l'accord de l'éducateur, de quitter un exercice, il doit cliquer sur le bouton **Sortie** qui se trouve en haut à gauche de chaque exercice et une boîte de dialogue apparaît pour lui demander confirmation et pour lui signaler que son travail va être enregistré.

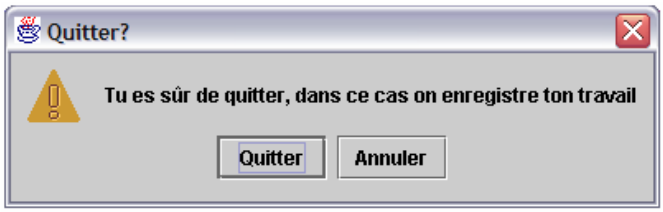


Figure 8 : Quitter un exercice.

Cette sauvegarde permet à l'élève de refaire un exercice sans répondre aux questions sur lesquelles il a déjà réfléchi. Mais, l'élève a la possibilité de recommencer un exercice qu'il a déjà fait.

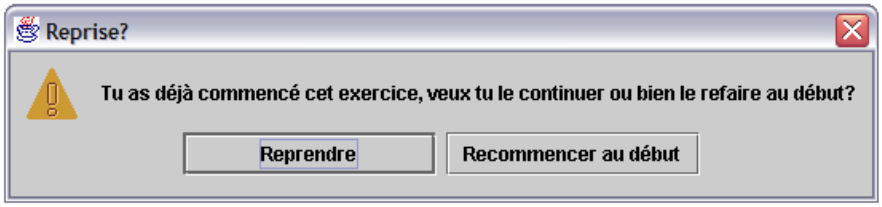


Figure 9 : Reprendre un exercice

2.4.2. L'éducateur

Toutes les actions que peut effectuer un éducateur ne sont possibles qu'à partir du menu ***Educateur***. L'éducateur doit alors entrer un mot de passe, ce qui empêche les élèves d'accéder à ces fonctionnalités.

Scénario 1 : L'éducateur crée un exercice du type **Orthographe**

Pour créer un exercice, l'éducateur doit se rendre dans le menu ***Educateur***, sélectionner ***Créer un nouvel exercice*** et choisir la catégorie de l'exercice qu'il veut concevoir. Pour un exercice du type **Orthographe**, l'éducateur doit d'abord indiquer le nombre de cases (de propositions) qu'il désire. Il en faut deux au minimum. Il doit aussi donner un titre à l'exercice. En cliquant sur le bouton **OK**, il verra apparaître le nombre de cases que il a demandé et dans lesquelles il pourra écrire les mots souhaités. Il doit indiquer la bonne réponse en cochant la case qui correspond à cette réponse. Il faut aussi écrire l'énoncé de la question. Pour confirmer la question, l'éducateur doit cliquer sur le bouton **Valider la question**. Il pourra alors créer une nouvelle question. Pour achever la création d'un exercice, il faut cliquer sur le bouton **Valider l'exercice**.

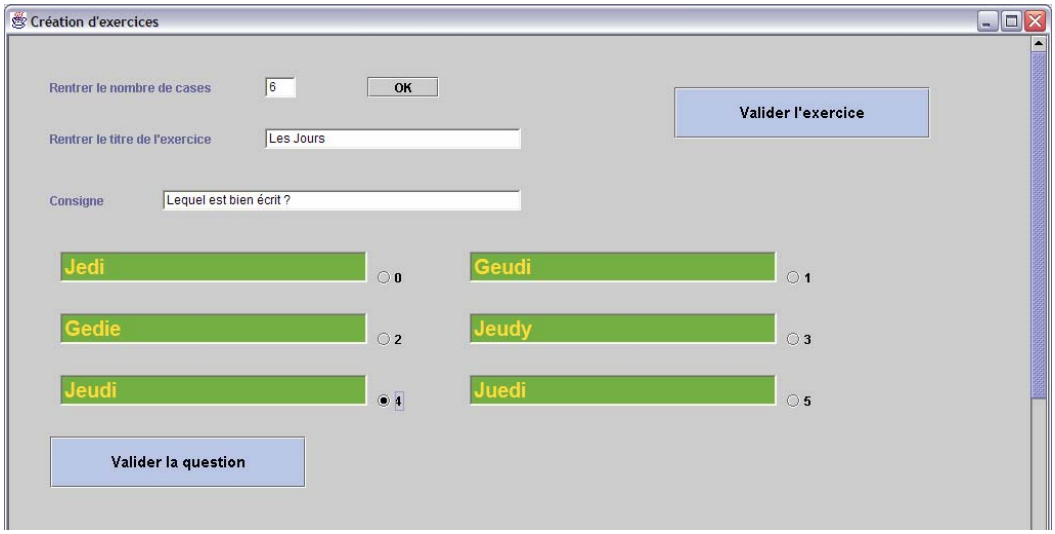


*Figure 10 : Création d'un exercice de la catégorie **Orthographe**.*

Scénario 2 : L'éducateur crée un exercice du type **Reconnaissance d'image**

Pour concevoir un exercice de la catégorie **Reconnaissance d'image**, l'éducateur doit d'abord signaler le niveau de l'exercice. Pour choisir l'image, il faut cliquer sur le bouton **Choisir l'image**. Ensuite, il doit indiquer le nombre de propositions qui seront fournies à l'élève (sauf s'il s'agit du niveau difficile). Lorsqu'il clique sur **OK**, l'image apparaît ainsi qu'un tableau dans lequel l'éducateur doit écrire les différentes solutions et une case où il faut inscrire la bonne réponse. Il faut également fournir un titre à l'exercice et un énoncé pour chaque question. La validation d'une question et de l'exercice s'effectue comme précédemment grâce aux boutons **Valider la question** et **Valider l'exercice.**

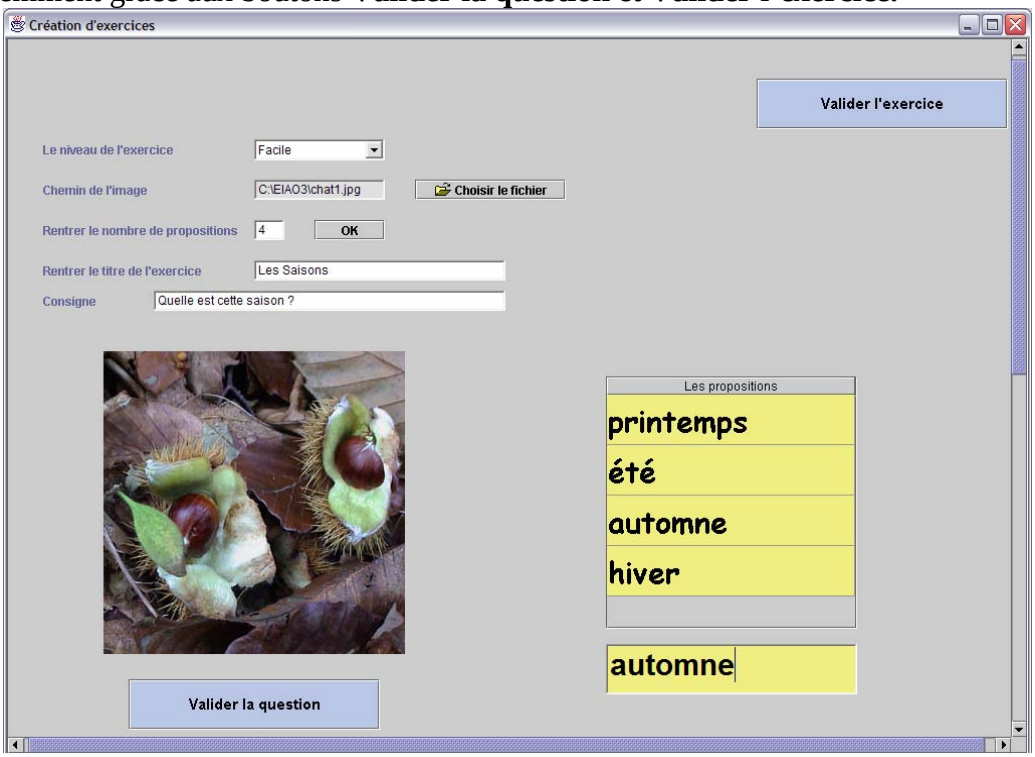


*Figure 11 : Création d'un exercice de la catégorie **Reconnaissance d'images.***

Scénario 3 : L'éducateur crée un exercice du type **Repérage**

Pour les exercices du type ***Repèrage***, l'éducateur doit commencer par choisir le niveau de l'exercice. Ensuite, il doit rentrer le nombre de cases. Celui-ci équivaut au nombre total de mots (affichés ou à deviner) que il veut mettre en place. Comme pour la catégorie précédente, le nombre de propositions correspond aux différentes solutions qui seront au service de l'élève. Une fois que l'éducateur a cliqué sur **OK**, les cases ainsi qu'un tableau apparaissent. Dans les cases, il doit écrire les mots et celles qui sont cochées vont correspondre aux mots que l'élève devra trouver. Dans le tableau, il faut écrire les réponses possibles en pensant à mettre celles qui sont correctes. Il faut aussi donner un titre à l'exercice et un énoncé à chaque question. La validation se fait comme pour les autres catégories.

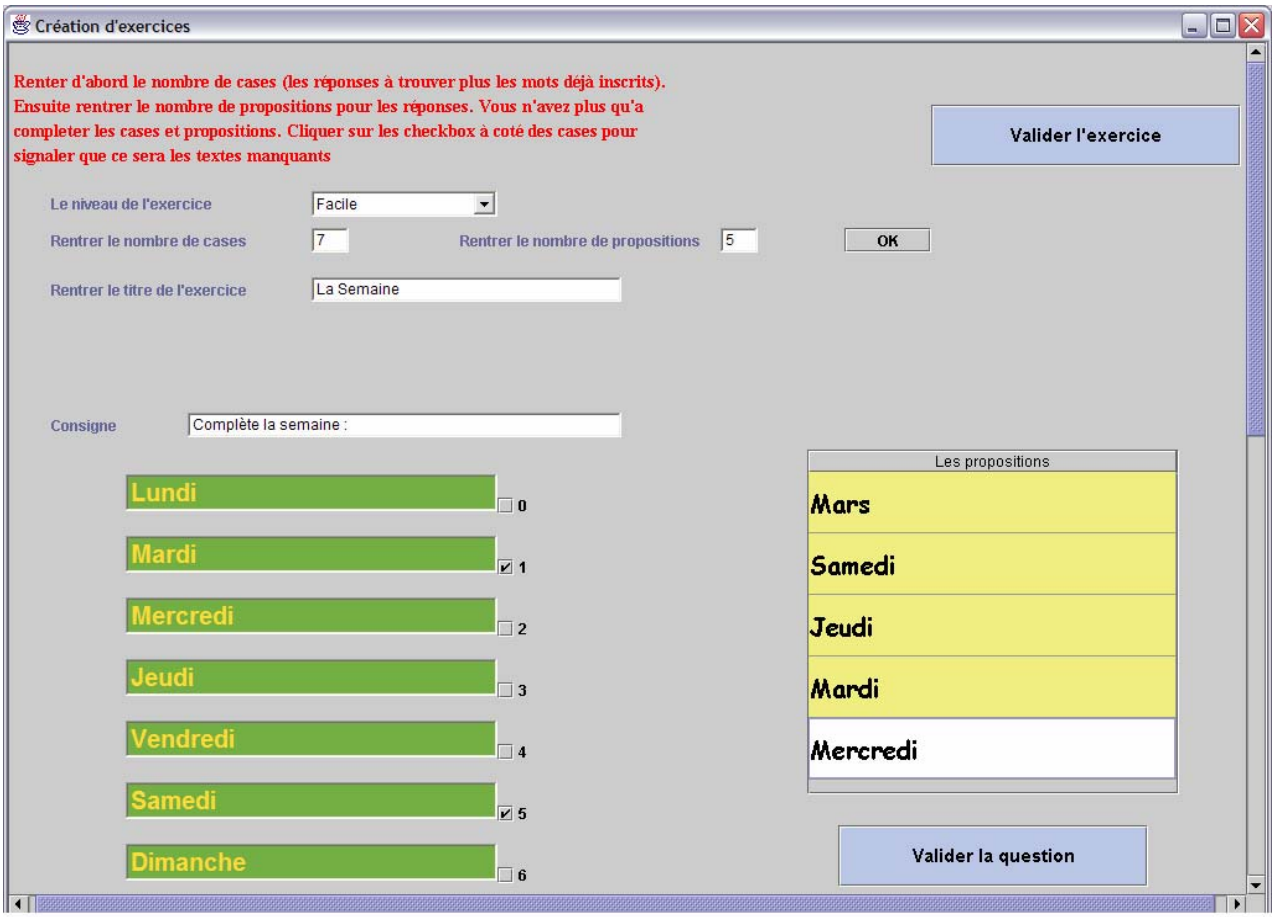


Figure 12 : Création d'un exercice de la catégorie ***Repérage***

Scénario 4 : L'éducateur supprime un exercice

Pour supprimer un exercice, l'éducateur doit aller dans le menu ***Educateur***, rubrique ***Supprimer un exercice***. Il se retrouve alors sur une page qui ressemble à la page d'accueil. Ensuite, il doit choisir l'exercice que il veut effacer dans les listes. Lorsque il clique sur le bouton où se trouve l'image, une fenêtre apparaîtra pour vous demander confirmation. Si vous choisissez **OUI**, l'exercice sera alors supprimer.

Figure 13 : Suppression d'un exercice

Scénario 5 : L'éducateur évalue un élève

Pour évaluer un élève, l'éducateur doit aller dans le menu ***Editer*** et sélectionner ***Editer la session***. Cela lui permet de voir dans quelle catégorie l'élève a travaillé, sur quels exercices, à quel niveau et si les exercices ont été terminés ou pas. Il peut aussi observer le taux de réussite de l'élève par exercice. Ce pourcentage correspond au rapport du nombre de questions sur lesquelles l'élève a réfléchi sur le nombre de réponses qu'il a fourni.

Figure 14 : Evaluation

Scénario 5 : L'éducateur supprime le fichier d'un élève

Lorsqu'un élève quitte l'IMP, les éducateurs peuvent effacer le fichier qui lui était réservé. Pour cela, il faut se rendre dans le menu ***Educateur*** et choisir la rubrique ***Supprimer un élève***.

3. Modélisation objet

Après avoir étudié tous les besoins de notre logiciel on va pouvoir modéliser notre problème dans un monde objet. On retrouvera :

Les objets qui vont être utilisés principalement par l'élève :

Les exercices, qui se distingueront par trois types :
- Les exercices sur l'orthographe.
- Les exercices sur le repérage dans le temps.
- Les exercices sur la reconnaissance d'images.

Chaque exercice d'un type pourra être utilisé dans différents niveaux.
Pour les exercices sur l'orthographe on aura un exercice facile et un difficile.
Pour les exercices sur le repérage on aura un niveau facile, intermédiaire et un difficile.
Pour les exercices sur la reconnaissance on aura aussi trois niveaux.

L'espion qui permet d'éditer les résultats de l'élève

Les objets utilisés par l'éducateur :

L'éducateur pourra créer des exercices du même type que les exercices de l'élève avec les mêmes niveaux : un objet par type d'exercice.
Chaque exercice qui est créé utilise une certaine structure selon le type (objet StructureExercice).
L'éducateur pourra supprimer des exercices : un objet SupprimeExercice
L'éducateur pourra supprimer un utilisateur : un objet SupprimeUtilisateur
L'éducateur pourra lui aussi utiliser l'espion.

La relation entre ces objets, expliquée sommairement ici, se retrouve dans le diagramme de classe ci-dessous ainsi que dans la description des classes.

3.2. Description des différentes classes

De nombreux attributs sont présents dans les classes et on énoncera uniquement ceux qui ont un intérêt pour bien comprendre notre travail. Les informations sur les classes sont aussi disponibles avec la commande javadoc qui génère les fichiers html et qui crée ainsi une API pour nos classes. Nos sources sont disponibles à l'adresse suivante : http://www.info.unicaen.fr/~jbossche/educatemps.
On utilise dans chaque classe des méthodes pour lire et écrire dans des fichiers textes (qui constituent notre base de données). On ne décrira pas dans cette partie notre manière de procéder pour la gestion de notre base car on en parlera en détail dans le chapitre « Gestion de la base de données ».
En ce qui concerne la partie graphique on utilise la librairie javax.swing.
Plusieurs couches sont à distinguer dans nos interfaces :
Tous nos composants sont mis dans des JPanels qui sont mis dans des JScrollPane et qui sont eux-mêmes mis dans une JFrame.

La classe Eleve

Elle permet de créer un élève quand un utilisateur ouvre une session.

Attributs

String nomUtlisateur qui contiendra le nom de l'élève qui a ouvert une session

Méthodes

public void modifieHistorique(String cle, String valeur) qui permet de modifier dans le fichier contenant les données sur un joueur un ensemble de clés/valeurs. On l'utilise pour modifier le numéro de question où l'élève s'est arrêté dans un exercice.

public String litHistorique(String cle, String valeur)
Cette méthode permet de renvoyer la valeur d'une clé. On l'utilise dans le cas de la reprise d'un exercice.

La classe Exercice

Cette classe permet de créer un exercice quelque soit son type, elle permet de définir un héritage de comportement et de structure pour les classes modélisant les autres exercices. Les classes dérivées de cette classe utiliseront ses attributs et ses méthodes.

Attributs

String urlExercice, l'url du fichier de donnée de l'exercice
Espion mouchard, l'espion qui enregistre les résultats de l'élève
Eleve etudiant, l'élève qui travaille sur l'exercice

Le constructeur

public Exercice(String url, Espion e, Eleve etu)

Les méthodes

Ce sont des méthodes d'accès aux attributs :

public Eleve getEleve() renvoie l'élève qui travaille sur cet exercice.
public Espion getEspion renvoie l'espion de l'exercice
public String getUrl() renvoie l'url du fichier de données de l'exercice. L'url peut être absolu ou relatif.

Les méthodes *reprise(), inter(int num) et smile() et reader()* ne contiennent rien et seront implémentées dans les classes héritières. Elles sont présentes pour bien définir le comportement des objets entre eux. Les attributs manipulés par ses méthodes étant définis dans les classes héritières, on peut juste donner leurs signatures.

public void reader() qui permet de lire le fichier de données d'un exercice et de modéliser chaque question. Quand on arrive à la fin de chaque question dans le fichier texte de données on appelle le constructeur d'une question pour le type d'exercice voulu avec les données lues en arguments. Une fois la lecture du fichier terminée on met chaque question dans un vecteur. (Cf. gestion de la base de données).

public void reprise() est une méthode qui permet de reprendre un exercice s'il a déjà été commencé.
On appelle la méthode *litHistorique(cle, valeur*) de la classe *Eleve* avec comme clé la nom de l'exercice. Si cette clé existe dans le fichier de l'élève on renvoie la valeur correspondante et on appelle la méthode *inter(int)* ci-dessous pour commencer à la question où l'on s'est arrêté.

public void inter(int num) permet de lire une question de l'exercice et de l'afficher à l'écran. On enlève alors les composants des *JPanel* et on en ajoute de nouveaux (nouvelles propositions, nouvelle consigne). C'est cette méthode qui géré les événements sur le bouton de validation de réponse.
Elle regarde la validité de la réponse choisie. Si la réponse est correcte elle se rappelle avec le numéro de la question suivante. Elle incrémente le compteur de clicks. Elle regarde si on a fini l'exercice. Elle demande à l'espion d'enregistrer des données. Elle gère la sortie de l'exercice.

public void smile() qui permet de changer l'état du smiley en fonction du taux de réussite à la question.

La classe ExerciceReperage

Elle hérite de la classe exercice et permet de gérer des exercices traitant le repérage.
Dans cette classe on redéfinit la méthode *reader()* qui permet de lire le fichier de données de l'exercice.
On appelle la méthode *getUrl()* de la classe mère *Exercice* et on lit le fichier de données. Quand on a lu une question on fait appel au constructeur *StructExoReperage* qui permet de modéliser une question d'un exercice de ce type. Chaque question est mise dans un vecteur qu'on pourra parcourir par la suite. Cette classe nous permet donc de modéliser l'ensemble de questions/réponses d'un exercice de type repérage.

Attributs

String niveau est le niveau de l'exercice (facile, intermédiaire, difficile)
int nbQuestion est le nombre de questions de l'exercice
Vector[] donnees_vide : pour chaque question on insérera dans le vecteur correspondant au numéro de question les numéros des cases réponses.
Vector[] donnees_propositions contient pour chaque question un vecteur avec l'ensemble des propositions de réponses
Vector[] donnees_label contient pour chaque question un vecteur contenant les mots à écrire (par exemple dans le cas d'un texte à trous).
String consigne est la consigne de l'exercice
int nbCases le nombre de cases totales (cases réponses + cases déjà remplies)
String[] donnees_reponses contient les réponses pour une question
int nbCasesVides le nombres de cases qui sont vides (les emplacements pour les réponses)
Vector tabQuestion contient l'ensemble des questions. Une question est représentée par un objet de la classe *StructExoReperage.*
String nomExercice le nom de l'exercice

Constructeur

public ExerciceReperage(String url, Espion e, Eleve etu)
Il appelle le constructeur de la classe Exercice par le mot clé super.

Méthodes

Les méthodes *reprise(), inter(int) et smile()* ne sont pas redéfinies

public void reprise()

public void inter(int num)

public void smile()

public void reader()

La classe ExoReperage1

Elle hérite de la classe *ExerciceReperage* et permet de gérer l'affichage d'un exercice de type repérage et de niveau facile.
Dans ce niveau on utilise comme composants graphiques des objets de type *DNDList* qui permettent de gérer les « drags and drops » (click sur un élément et déplacement à un autre endroit).
La liste de propositions et les emplacements des réponses sont des objets de ce type.
On utilise des objets de type *JButton* pour la validation d'une question, pour la sortie d'un exercice. A chaque bouton est relié un *Listene*r qui permet de capter le click sur le bouton et donc de déclencher un événement en conséquence.

Attributs

Elle reprend ceux de la classe mère. On définit des attributs pour les composants graphiques que l'on énoncera pas ici (trop lourd et sans grand intérêt).
On a ajouté des attributs *int nbClick, int nbClickTotal, int reussite* qui permettent d'évaluer l'élève à chaque question et après avoir arrêté l'exercice. Ces valeurs sont communiquées à la classe *Espion*.
L'attribut *int numQuestion* permet de connaître le numéro de la question en cours.

Métodes

C'est dans cette classe que l'on va redéfinir les méthodes *reprise(), inter(int) et smile().*

public void reprise()

public void inter(int num)

public void smile()

La classe ExoReperage2

Cette classe permet de travailler sur un exercice de repérage de niveau intermédiaire. Dans ce niveau l'élève doit lui-même écrire les réponses en s'inspirant des différentes propositions. On a plus d'objets *DNDList* qui gère les « drags and drops » mais uniquement des *JTable* contenant les propositions. Les champs pour les réponses sont des *JTextField.*

C'est dans cette classe que l'on va redéfinir les méthodes reprise(), inter(int) et smile().

La classe ExoReperage3

Cette classe permet de travailler sur un exercice de repérage de niveau difficile. Dans ce niveau l'élève n'a plus d'aide (pas de proposition). On a plus de *JTable* mais uniquement des *JLabel* et des *JTextField.* On utilise quand même la méthode *reader()* de la classe mère malgré qu'il n'y ai pas de proposition. Quant on crée un exercice de ce type et de niveau difficile on enregistrera, dans le fichier texte, au niveau de la ligne proposition, un blanc. Le vecteur de propositions sera donc nul. On a donc choisit de garder quelque soit le niveau d'exercice la même structure de fichier texte.

C'est dans cette classe que l'on va redéfinir les méthodes *reprise(), inter(int) et smile().*

La classe ExerciceOrth

Elle hérite de la classe *Exercice* et permet de gérer des exercices traitant l'orthographe des mots.
Dans cette classe on redéfinit la méthode *reader()* qui permet de lire le fichier de données de l'exercice.

On appelle la méthode *getUrl()* de la classe mère Exercice et on lit le fichier de données. Quand on a lu une question on fait appel au constructeur *StructExoOrth* qui permet de modéliser une question d'un exercice de ce type. Chaque question est mise dans un vecteur qu'on pourra parcourir par la suite. Cette classe nous permet donc de modéliser l'ensemble de questions/réponses d'un exercice de type orthographe.

Attributs

String niveau, le niveau de l'exercice
int nbQuestion, le nombre de questions de l'exercice
Vector[] structure, contenant les propositions
String question, la consigne de l'exercice
String reponse, la bonne réponse
Vector tabQuestion est un vecteur contenant l'ensemble des questions de l'exercice (objet *StructExoOrth*)
String nomExercice, le nom de l'exercice

Constructeur

public ExerciceOrth(String url, Espion e, Eleve etu) appelle le constructeur de la classe mère Exercice.

Méthodes :

Les méthodes *reprise(), inter(int) et smile()* ne sont pas redéfinies

public void reprise()

public void inter(int num)

public void smile()

public void reader()

La classe ExoOrth1

Elle hérite de la classe *ExerciceOrth* et permet de gérer l'affichage d'un exercice sur l'orthographe de niveau facile. L'élève se trouve devant plusieurs propositions d'orthographe pour un même mot et doit cliquer sur celui bien orthographié. Dans ce niveau on utilise comme composants graphiques des *JButton* pour les propositions et des *JLabel* pour le titre et la question.
On a choisit des *JButton* car il est facile de détecter un click sur un bouton.
A chaque bouton est relié un *Listener* qui permet de capter le click sur le bouton et donc de déclencher un événement en conséquence. Ici les clicks sur les boutons de propositions créent un événement déclenchant une action qui vérifie la validité de la réponse.

Constructeur

public ExoOrth1(String url, Espion e, Eleve etu) qui appelle le constructeur de la classe *ExerciceOrth*

Attributs

Elle reprend ceux de la classe mère. On définit des attributs pour les composants graphiques.
On a ajouté des attributs *int nbClick, int nbClickTotal, int reussite* qui permettent d'évaluer l'élève à chaque question et après avoir arrêté l'exercice. Ces valeurs sont communiquées à la classe Espion.
L'attribut *int numQuestion* permet de connaître le numéro de la question en cours.

Méthodes

C'est dans cette classe que l'on va redéfinir les méthodes *reprise(), inter(int) et smile().*

public void reprise()
public void inter(int)
public void smile()

La classe ExoOrth2

Elle permet de faire un exercice sur l'orthographe en niveau difficile. Dans ce niveau l'élève regarde les différentes propositions sur l'orthographe d'un mot et doit recopier ce mot dans un champ prévu à cet effet.
Les attributs pour les propositions ne sont plus des *JButton* mais des *JLabel.* Le champ de réponse est un *JTextField.*
C'est dans cette classe que l'on va redéfinir les méthodes *reprise(), inter(int) et smile().*

La classe ExerciceReconnaissance

Elle hérite de la classe *Exercice* et permet de gérer des exercices traitant la reconnaissance d'une image.
Dans cette classe on redéfinit la méthode *reader()* qui permet de lire le fichier de données de l'exercice.
On appelle la méthode *getUrl()* de la classe mère Exercice et on lit le fichier de données. Quand on a lu une question on fait appel au constructeur *StructExoReconnaissanc*e qui permet de modéliser une question d'un exercice de ce type. Chaque question est mise dans un vecteur qu'on pourra parcourir par la suite. Cette classe nous permet donc de modéliser l'ensemble de questions/réponses d'un exercice de type reconnaissance.

Attributs :

String reponse : la réponse
int nbQuestion : le nombre de questions de l'exercice
Vector[] donnees_propositions : l'ensemble des propositions
Vector tabQuestion : l'ensemble des questions (vecteur contenant des objets de type *StructExoReconnaissance*)

String nomExercice : le nom de l'exercice
String niveau : le niveau de l'exercice
String consigne : la consigne
String urlImage : le chemin de l'image (relatif ou absolu)

Constructeur

public ExerciceReconnaissance(String url, Espion e, Eleve etu) qui appelle le constructeur de la classe mère Exercice.

Méthodes

Les méthodes *reprise(), inter(int) et smile()* ne sont pas redéfinies

public void reprise()

public void inter(int num)

public void smile()

public void reader()

La classe ExoReconnaissance1

Elle permet de travailler avec un exercice de reconnaissance avec image de niveau facile.
Elle hérite de la classe *ExerciceReconnaissance*.
A ce niveau l'élève doit faire glisser la bonne proposition qui identifie l'image sur un champ réponse.
On utilise donc des objets graphiques *DNDList* qui gère le « drag and drop ».
Pour la validation de la réponse on a un *JButton.*

Constructeur

public ExoReconnaissance1(String url, Espion e, Eleve etu) qui appelle le constructeur de la classe *ExerciceReconnaissance*

Attributs

Elle reprend ceux de la classe mère. On définit des attributs pour les composants graphiques (*DNDList* pour les propositions et le champ réponse, *JLabel* pour le titre et la consigne et *JButton* pour la validation).
On a ajouté des attributs *int nbClick, int nbClickTotal, int reussite* qui permettent d'évaluer l'élève à chaque question et après avoir arrêté l'exercice. Ces valeurs sont communiquées à la classe *Espion*.
L'attribut *int numQuestion* permet de connaître le numéro de la question en cours.

Méthodes

C'est dans cette classe que l'on va redéfinir les méthodes *reprise(), inter(int) et smile().*

public void reprise()
public void inter(int)
public void smile()

La classe ExoReconnaissance2

Cette classe hérite de la classe *ExerciceReconnaissance*. Elle permet de gérer un exercice de type reconnaissance de niveau intermédiaire. A ce niveau on a toujours une liste de propositions mais on n'a plus de composant *DNDList* car il n'y a plus de système de « drags and drops »

Méthodes

C'est dans cette classe que l'on va redéfinir les méthodes reprise*(), inter(int) et smile().*

public void reprise()
public void inter(int)
public void smile()

La classe ExoReconnaissance3

Cette classe hérite de la classe *ExerciceReconnaissance*. Elle permet de gérer un exercice de type reconnaissance de niveau difficile. A ce niveau on a plus de listes de propositions. On a juste un *JLabel* contenant l'image et un *JTextField* pour saisir la réponse.

Méthodes

C'est dans cette classe que l'on va redéfinir les méthodes *reprise(), inter(int) et smile().*

public void reprise()
public void inter(int)
public void smile()

La classe StructExoReperage

Cette classe permet de définir la structure d'une question d'un exercice sur le repérage.
Un exercice, au niveau données questions/réponses, sera composé de plusieurs objets de ce type.

Attributs

String consigne : la consigne de la question
int nbCases : le nombre de cases totales
int nbCasesVides : le nombre de cases réponses
Vector donnees_vide : les numéros des cases réponses

Vector donnees_label : les valeurs des champs déjà remplies
Vector donnees_propositions : les propositions
String[] donnees_reponses : les réponses

Constructeur

public StructExoReperage1(String consigne,int nbCases,int nbCasesVides,Vector donnees_vide, Vector donnees_label, Vector donnees_propositions,String[] donnees_reponses)

Méthodes

Ce sont de méthodes d'accès aux attributs de la classe

public int getNbCases()
public Vector getDonnees_label()
public Vector getDonnees_propositions()
public Vector getDonnees_vide()
public String getConsigne()
public String[] getDonnees_reponses()
public int getNbCasesVides()

La classe StructExoOrth

Cette classe permet de définir la structure d'une question d'un exercice sur l'orthographe.
Un exercice sur l'orthographe, au niveau données questions/réponses, sera composé de plusieurs objets de ce type.

Attributs

Vector donnees : l'ensemble des propositions
String reponse : la bonne réponse
String question : la question

Constructeur

public StructExoOrth1(String question, String reponse, Vector donnees_question)

Méthodes

Ce sont des méthodes d'accès aux attributs :

public Vector getDonnees_question()
public String getReponse()

public String getQuestion()

La classe StructExoReconnaissance

Elle permet de définir la structure d'une question sur un exercice de type reconnaissance d'images.

Attributs

String consigne : la consigne de la question
Vector donnees_propositions : l'ensemble des propositions
String reponse : la réponse
String urlImage : le chemin de l'image

Constructeur

public StructExoReconnaissance1(String consigne,String urlImage,String reponse, Vector donnees_propositions)

Méthodes

Ce sont des méthodes d'accès aux attributs

public String getUrlImage()
public String getReponse()
public Vector getDonnees_propositions()
public String getConsigne()

La classe Espion

Cette classe permet d'enregistrer les résultats de l'élève sur une session. Cet espion suit l'élève sur chaque exercice et enregistre sa réussite. L'espion écrit les données dans un fichier. On peut alors le consulter à chaque moment de la session.

Attributs

Les attributs sont des objets graphiques qui vont permettre d'éditer le fichier espion à l'écran (*JLabel, ImageIcon*…)

Constructeur

public Espion(String nomUtilisateur) élimine le fichier espion.txt s'il existe et en crée un nouveau. Il insère le nom de l'élève dans ce fichier. Pour gérer le fichier texte on utilise les objets *Properties* (Cf. gestion de la base de données).

Méthodes

public void insertData(String cleExo, String nomExo,int niveau, int reussite, int fini) qui permet d'insérer le taux de réussite dans le fichier espion. Les informations écrites dans le fichier sont le titre de l'exercice, le niveau, le taux de réussite, et une information permettant de savoir si l'exercice est terminé ou non.

public void edite() permet d'éditer les résultat est de créer tous les composants graphiques nécessaires pour insérer toutes les informations du fichier espion.

public void montre() permet d'afficher, d'arranger tous les composants graphiques créés.

La classe SupprimeExo

Cette classe permet de supprimer un exercice et plus précisément son fichier de données.
Seul l'éducateur utilisera cette classe. On retrouve trois listes pour les différents types d'exercices avec un bouton « supprimer » par liste.

Attributs

Les attributs sont surtout des objets graphiques qui vont permettrent d'afficher la liste des exercices.
Educatemps i est la fenêtre principale à modifier une fois le fichier supprimé.

Constructeur

public SupprimeExo(Educatemps i)
Le constructeur prend un Objet *Educatemps* en paramètre car une fois l'exercice supprimé on doit enlever cette exercice des listes d'exercices de la fenêtre principale du logiciel. On appelle la méthode *creeListeExo()* de la classe *Educatemps.*

Méthodes

public void supprime() qui permet de supprimer le fichier de données d'un exercice. Le bouton qui a été cliqué déclenche un événement. Selon le bouton enfoncé (suppression pour l'orthographe, ou le repérage…), on récupérera l'information sur l'Item sélectionné dans la liste correspondante et on pourra supprimer le fichier.

public void changeDonneesUtilisateur(String nomCleExo) permet une fois l'exercice supprimé de parcourir notre base de données utilisateur et d'enlever, si la clé (nom de l'exercice) existe, la ligne correspondant à cette clé pour chaque utilisateur.

La classe SupprimeUtilisateur

Cette classe permet de la suppression d'un utilisateur, et plus précisément la suppression de son fichier de données.

Attributs

Les attributs sont des attributs graphiques permettant l'affichage de la liste des utilisateurs.
On vérifie que l'utilisateur que l'on souhaite supprimer n'a pas de session ouverte.

Constructeur

public SupprimeUtilisateur(String nom) : le paramètre nom est le nom de celui qui a une session en cours. Il crée une fenêtre avec une un bouton pour la suppression et une liste déroulante pour l'instant vide qui contiendra les utilisateurs. Le bouton pour la suppression, lorsqu'il sera enfoncé, affichera un message de confirmation de suppression. Une fois la suppression faite on appelle la méthode *creeListeUtilisateur* pour réinitialiser la liste des utilisateurs.

Méthodes

public void creeListeUtilisateur() permet de créer la liste des utilisateurs en parcourant tous les fichiers textes de notre base de données et en ajoutant dans une liste le nom et prénom de l'élève.

La classe Aide

Cette classe permet à l'élève ou à l'utilisateur de fournir des renseignements sur le fonctionnement du logiciel et de ses exercices.

La classe creeExercice

Cette classe va permettre à l'éducateur de créer un exercice.
C'est une classe abstraite qui permet de faire un héritage de comportement.
On donne plus de détails sur l'écriture des données dans un fichier dans la partie « gestion de la base de données »

Méthodes

public abstract void inter() elle permettra de créer une fenêtre qui accueillera tous les composants graphiques nécessaires à la saisie des questions.

public abstract void valideCase() permettra de valider si les informations nécessaires à la création des masques de saisie pour les questions sont correctes.

public abstract void valideQuestion() permettra de savoir si les informations rentrées dans les masques de saisie sont correctes. Dans ce cas on enregistrera la question et on supprimera les masques de saisies. Chaque question est enregistrée dans un vecteur sous forme de chaîne de caractère.

public abstract void valideExo() permet de valider l'exercice. Ce bouton est accessible quand une question a été validée.

La classe creeExoReperage

Elle implémente la classe abstraite *creeExercice* et donc les méthodes abstraites de la classe *creeExercice*. Les attributs sont des attributs essentiellement graphiques.

Méthodes

public void valideCase()

public void valideQuestion()

public void valideExo()

public void inter()

La classe CreeExoReconnaissance

Elle implémente la classe abstraite *creeExercice* et donc les méthodes abstraites de la classe *creeExercice*. Les attributs sont des attributs essentiellement graphiques.

Méthodes

public void valideCase()

public void valideQuestion()

public void valideExo()

public void inter()

La classe CreeExoOrth

Elle implémente la classe abstraite *creeExercice* et donc les méthodes abstraites de la classe *creeExercice*. Les attributs sont des attributs essentiellement graphiques.

Méthodes

public void valideCase()

public void valideQuestion()

public void valideExo()

public void inter()

La classe PropsHandler

Les *Properties* sont des objets qui permettent de gérer un ensemble de clés/valeurs.
Cette classe permet de créer un objet *Properties* à partir d'un fichier texte. On pourra modifier une valeur correspondant à une clé et ensuite écrire dans le fichier texte nos nouvelles données.

Constructeur

public PropsHandler()

Méthodes

public Properties loadProps(String filePath) qui renvoie une *Properties* avec l'ensemble clés/valeurs contenues dans le fichier texte de chemin filePath.

public boolean storeProps(String filePath, Properties newProps) qui permet de réécrire le nouvel ensemble clés/valeurs (la nouvelle *Properties*) dans un fichier. La méthode de la classe *Properties store(OutputStream out, String header)* permet d'écrire un ensemble de clés/valeurs dans un flot de sortie. Header étant l'en-tête à insérer au fichier.
Les méthodes *put(String cle, String valeur)* et *setProperty(String cle, String valeur*) permettent respectivement d'insérer et de modifier un ensemble de clés/valeurs.

La classe Educatemps

Cette classe permet d'utiliser tous les objets qui viennent d'être définis. C'est la page principale du logiciel.

Attributs

Eleve etudiant : élève qui travaille sur le logiciel
JLabel labelUtilisateur : on placera le nom de l'élève
JComboBox listeOrth : liste comportant les exercices sur l'orthographe
JComboBox listeReperage : liste comportant les exercices sur le repérage
JComboBox listeReconnaissance : liste comportant les exercices sur la reconnaissance
JButton boutonOrth : permettra d'ouvrir l'exercice choisit sur l'orthographe
JButton boutonReperage : permettra d'ouvrir l'exercice choisit sur le repérage
JButton boutonReconnaissance : permettra d'ouvrir l'exercice choisit sur la reconnaissance
JMenuBar menubar : la barre de menu de la fenêtre
Dans cette barre on aura des *JMenuItem* : un pour « fichier », un pour « édition », un pour « éducateur » et un pour « aide ».
Dans le menu éducateur on aura des sous menus :

- Pour créer des exercices à l'intérieur duquel on aura un sous menu pour l'orthographe, un pour le repérage et un pour la reconnaissance)
- Pour supprimer des exercices

- Pour supprimer des élèves

Constructeur

public Educatemps () initialise tous les composant graphiques

Méthodes

public void creeListeExo() qui permet de remplir les listes déroulantes comportant les exercices

public void popup() qui permet d'ouvrir une fenêtre pour pouvoir ouvrir ou changer de session.
Cette méthode crée une fenêtre avec d'un côté la possibilité de s'enregistrer en tant que nouvel utilisateur et d'un autre côté la possibilité de choisir son nom et prénom dans une liste pour ouvrir sa session.

public void creeProfil(String nom, String prenom) qui permet d'insérer un nouvel utilisateur dans notre base de données. Cette méthode est appelée quand l'utilisateur crée un compte dans la méthode *popup().*

public void creeListeUtilisateur() qui permet de remplir la liste déroulante des utilisateurs quand on ouvre une session. Cette méthode est appelée dans la méthode *popup*. On parcourt l'ensemble de nos fichiers contenu dans le répertoire usersdata, on lit chaque fichier, on cherche les valeurs correspondant aux clés NOM et PRENOM puis on ajoute ces valeurs dans la liste.

4. Modèle dynamiques : trace des événements

Initialisation

L'élève lance le logiciel

La méthode *popup()* de la classe *EducaTemps* est appelée :

Educatemps bloque les menus et boutons d'accès aux exercices.

Educatemps parcours la liste des fichiers utilisateurs dans le répertoire usersdata. Pour chaque fichier il lit la clé NOM et la clé PRENOM (avec les *Properties*) et insére ces données dans la liste des utilisateurs.

Si l'utilisateur est nouveau alors ils doit remplir les deux champs nom et prénom puis il valide.

Si un utilisateur de ce nom existe déjà on lui propose de saisir un autre nom et prénom ou bien de regarder s'il ne figure pas dans la liste des utilisateurs. La vérification pour éviter

les doublons est assez simple, on regarde s'il n'existe pas déjà un fichier nommé « NOM PRENOM.txt »

Si l'utilisateur n'existe pas, alors on crée un fichier texte nommé « NOM PRENOM.txt » dans le répertoire userdata puis on insère son nom et son prénom dans le fichier créé (avec les *Properties*). L'utilisateur n'est plus considéré comme nouveau et on ajoute son nom dans la liste des utilisateurs pour qu'il puisse ouvrir une session.

Si l'utilisateur est déjà enregistré, il choisit son nom et prénom dans la liste des utilisateurs et clique sur le bouton de validation. *Educatemps* crée alors un nouvel objet *Elève* pour la session en cours. Il crée aussi un *Espion* avec le nom et prénom de l'élève.

Une session est ouverte

Educatemps débloque les boutons d'accès aux exercices et aux menus.

L'élève choisit un exercice en choisissant son exercice dans la liste des exercices puis clique bouton de validation.

A ce moment là, *Educatemps* regarde le niveau de l'exercice choisit et appelle le constructeur de la bonne classe. Le niveau de l'exercice est visible dans un tableau. En effet quand on a créé la liste des exercices on a inséré le nom, l'url et le niveau de l'exercice dans un tableau de taille : nombre d'exercices*3.
De ce fait on connaît le niveau en faisant mesExosOrth[malisteOrth.getSelectedIndex()][2]
Si c'est un exercice de type reconnaissance le répertoire dans lequel se trouvera le fichier de données sera « exercice/reconnaissance ». Pour les exercices sur l'orthographe le chemin sera « exercice/orth » et pour le repérage il sera « exercice/reperage ».

L'exercice est créé, et *Educatemps* lance la méthode *reprise()* sur cet exercice.

L'exercice regarde si l'élève a déjà commencé cet exercice. L'exercice applique à l'élève sa méthode *litHistorique(nomExo).* On cherche en fait la clé nomExercice dans le fichier de l'élève et on retourne sa valeur.

Si la valeur de retour de *litHistorique* est nulle alors on commence l'exercice au début. L'exercice appelle sa méthode *inter(0).*

Si la valeur de retour n'est pas nulle alors l'exercice propose à l'élève de recommencer l'exercice au début ou bien de le continuer. Cette demande se fait par l'intermédiaire d'une *JOptionPane*. Si la valeur de retour est égale au nombre de question de l'exercice alors cela signifie que l'élève a terminé l'exercice. Dans ce cas l'exercice propose à l'élève de le refaire au début ou de quitter.
Dans le cas on l'élève choisit de sortir il retourne sur la page principale avec sa session toujours ouverte.

L'élève est dans un exercice à une question d'indice quelconque.

S'il répond correctement à la question demandée l'exercice ouvre une fenêtre avec un smiley et l'exercice augmente de 1 le compteur de clicks de l'élève. La méthode inter est appelée avec le numéro de la question suivante.

S'il répond correctement et que la question était la dernière, l'exercice agit de la même manière qu'au dessus mais enregistre le score de l'élève dans l'espion : appel de la méthode *insertData* de la classe *Espion*. Il modifie l'historique de l'élève : application de la méthode *modifieHistorique(nomExo, numeroQuestion).*

S'il ne répond pas correctement, l'exercice augmente le compteur de clicks de l'élève et indique à l'élève que ce n'est pas la bonne réponse (apparition d'une fenêtre avec un smiley pas content). L'exercice reste à la même question.

Si l'élève décide de sortir, l'exercice demande une confirmation à l'élève en ouvrant une boîte de dialogue. Si l'élève clique sur « annuler » l'exercice revient dans son état précédent. S'il clique sur « sortir » l'exercice applique à l'élève la méthode *modifieHistorique(nomExo, numeroQuestion)* pour enregistrer le numéro auquel l'élève s'est arrêté. L'exercice applique aussi à l'espion sa méthode *insertData* qui permettra d'insérer le taux de réussite sur les questions répondues. L'élève se retrouve sur la page principale.

L'élève veut regarder les résultats de sa session

Il clique alors sur le menu de la fenêtre principale. *Educatemps* lance la méthode *edite()* sur *l'espion*.
Une fenêtre apparaît avec les résultats de l'élève pour les différents exercices faits.

Un autre élève veut travailler

L'élève choisit dans le menu fichier « changer d'utilisateur », *Educatemps* lance le méthode *popup()* qui fait apparaître la fenêtre de session.

Quand l'éducateur doit créer, supprimer un exercice ou encore supprimer un utilisateur
Educatemps demande de rentrer un mot de passe par l'intermédiaire d'une boîte de dialogue. Si le mot de passe est mauvais on reste dans la fenêtre principale sinon on a accès aux actions.

L'éducateur veut créer un exercice

L'éducateur choisit dans le menu éducateur le type d'exercice à créer. En fonction du type d'exercice, *Educatemps* appel le constructeur d'une des trois classes *CreeExoOrth, CreeExoReperage, CreeExoReconnaissance. Educatemps* applique la méthode *inter()* à cet objet qui fait apparaître une fenêtre.
Selon le niveau de l'exercice que l'éducateur choisi on affichera ou non le champ concernant le nombre de propositions.

L'éducateur rentre les informations demandées pour créer une question (nombres de cases, url de l'image…) et clique sur le bouton « valider ».

CreeExercice regarde alors la validité des informations rentrées. Si elles sont fausses le logiciel propose de ressaisir les données sinon *CreeExercice* affiche les composants que

l'éducateur doit remplir (questions, propositions, réponses…). Il bloque la liste de choix pour les niveaux.
Une fois tous les champs remplis l'éducateur clique sur le bouton « valider la question ».

CreeExercice vérifie la validité des données et enregistre la question sous forme de chaîne de caractères, formatée selon notre modèle. On rajoute dans la chaîne « \n » si l'on souhaite passer à la ligne et entre chaque propositions on ajoute des « ; » par exemple. Cette chaîne est mise dans un vecteur.

CreeExercice retire tous les composants de la question et propose de ressaisir une autre question ou bien de valider l'exercice.

Si l'éducateur valide l'exercice *CreeExercice,* pour chaque élément du vecteur contenant les questions, écrit dans un fichier texte au nom de l'exercice la chaîne de caractère que contient chaque élément. Entre chaque élément, *CreeExercice* écrit la ligne « //// » dans le fichier pour séparer les questions. Une fois le fichier créé on revient à la fenêtre principale.

L'éducateur veut supprimer un élève
Il clique dans le menu éducateur sur « supprimer un utilisateur ». *Educatemps* appelle alors le constructeur de la classe *SupprimeUtilisateur* avec le nom de l'utilisateur en cours et applique à l'objet créé la méthode *creeListeUtilisateur*. Cette méthode a le même fonctionnement que lors de la création de la liste des utilisateurs de la méthode *popup()* de la classe *Educatemps*.
L'éducateur choisit un élève dans la liste des élèves et clique sur le bouton « supprimer ».
SupprimeUtilisateur vérifie que l'utilisateur que l'on souhaite supprimer n'est pas l'utilisateur de la session en cours. Si c'est le cas on propose renvoie un message d'alerte.
Si on peut supprimer l'élève, alors *SupprimeUtilisateur* efface le fichier portant le nom de l'élève sélectionné dans le répertoire « usersdata ». La liste des utilisateurs est mise à jour dans la fenêtre principale du logiciel et dans la fenêtre actuelle.

L'éducateur veut supprimer un exercice
On procède de la même manière que pour la suppression d'un utilisateur. Une fois le fichier effacé on parcourt, dans notre base de données, tous les utilisateurs et on regarde s'ils n'ont pas une trace de cet exercice. Si c'est le cas on supprime ces informations qui ne sont plus utilisables.
On met à jour la liste des exercices de la fenêtre principale.

L'élève ou l'éducateur à besoin d'aide : il clique sur le menu « aide » de la barre de menu de la fenêtre. Une fenêtre apparaît avec le manuel utilisateur du logiciel.

L'élève veut quitter : il clique sur la croix en au à gauche de la fenêtre ou bien il peut aller dans le menu « fichier » puis choisir « quitter ». A ce moment là, on demande une confirmation de sortie : « sortir », « annuler ». S'il choisit « annuler » on reste sur la page principale sinon le logiciel se termine.

5. Gestion de la base de données

Un grand nombre d'informations concernant l'élève et les exercices doivent être sauvegardés pour les réutiliser ensuite.
Pour ce faire nous avons choisi d'utiliser des fichiers textes pour stocker les informations. Nous n'avons pas choisit une base de données type SQL car il était plus difficile d'installer un serveur sur les machines de l'IMP.

Les informations des exercices

- Création des exercices

Quelques soit le type d'exercices les éducateurs peuvent créer leurs exercices via une interface administrateur. Pour cela ils doivent saisir les différentes propositions, les réponses possibles, insérer l'url d'une image, cocher la bonne réponse dans la liste de propositions …
Une fois les données rentrées ils valident la question et peuvent en créer une autre.
La question est insérée dans un vecteur qui contient la structure et les données de la question.
Quand l'éducateur a terminé la saisie des questions il doit valider l'exercice. Le vecteur contenant toutes les questions est alors parcouru et les données sont écrites dans un fichier texte au nom du titre de l'exercice. On utilise des *PrintStream* et donc des *OutputStream*.
Chaque donnée d'une question est séparée par un passage à la ligne dans le fichier. Pour la liste des propositions on sépare chaque proposition par des « ; » qui sont les séparateurs.
Le séparateur pour séparer chaque question est « ///// ».

Exemple de fichier :

```
Les saisons                          //Le titre de l'exercice
Les saisons_2.txt                    //Le nom du fichier
2                                    //le niveau de l'exercice
2                                    //le nombre de questions
A quel saison te fais penser cette image ?   //l'intitulé de la question
C:\EIAO3\bg-flocons.gif              //url de l'image
hiver                                //la réponse
été;automne;printemps;hiver;         //les propositions
////                                 //le séparateur entre question
A quel saison te fais penser cette image ?
C:\EIAO3\chat1.jpg
automne
printemps;automne;hiver;été;
////
```

- Lecture des données des exercices

Quand l'élève choisit un exercice on ouvre le fichier texte correspondant puis on le lit.
On utilise des *BufferReader* et *FileReader* pour lire les lignes du fichier. Quand une ligne contient plusieurs propositions on utilise des *StringTokenizer* pour obtenir chaque donnée.

A chaque fois qu'une question est lu, c'est-à-dire quand on tombe sur « //// » on appelle le constructeur de la classe *StructExoOrth* ou *StructExoReperage* ou *StructExoReconnaissanc*e selon le type d'exercice pour créer la structure de la question. On insère ces objets dans un vecteur.

Information élève et espion

Chaque élève a un fichier à son nom contenant diverses informations sur lui.
La structure de ce fichier est un peu différente de celle des exercices car on utilise des Objets *Properties*. Ces objets permettent, de part leurs méthodes, d'écrire, de lire dans un fichier texte un ensemble de couple (clé, valeur). Les données contenues dans le fichier d'un élève étant principalement des informations sur les exercices commencés, il est facile de retrouver un exercice et de savoir s'il l'a déjà commencé. Trouver une clé, ou une valeur est très facile.
On a créé une classe *PropsHandler* pour gérer l'insertion et la mise à jour de clés/valeurs.
Cette classe permet de charger un fichier texte pour voir ses couples clés/valeurs. Il est aussi possible de modifier l'ensemble clés/valeurs d'une *Properties* et de la réinsérer dans le fichier texte. Une *Properties* est une structure, elle peut être assimilée à une structure de table en SQL et le fichier texte que l'on charge est le fichier de données.

Exemple de fichiers :

```
#Fichier de config pour utilisateur ...
#Thu May 22 10:55:18 CEST 2003
NOM=Durand
PRENOM=Paul
```

Requête :

```
PropsHandler handler = new PropsHandler();          //crée une Properties vides
String propertiesFilePath = "C:/monfichier.txt";
Properties props = handler.loadProps(propertiesFilePath);     //charge les données du fichier
props.put("EXERCICE1","4");                                    //insére une nouvelle clé et sa valeur
props.setProperty("NOM","Dupond");                             //modifie une valeur suivant une clé
handler.storeProps(propertiesFilePath,props);      //réécrit les nouvelles données dans le fichier
```

Fichier après la requête :

```
#Fichier de config pour utilisateur ...
#Thu May 22 10:58:08 CEST 2003
NOM=Dupond
PRENOM=Paul
EXERCICE1=4
```

6. Conclusion

Avec **ECUCATEMPS**, nous pensons avoir répondu à la majorité des objectifs fixés. Tout d'abord, notre interface graphique, avec ses gros caractères, ses couleurs, donne au logiciel une bonne lisibilité, ce qui facilite le travail de l'élève. Ensuite, chaque session peut être éditée grâce à la classe *Espion*. L'éducateur peut donc évaluer l'élève et celui-ci a alors un suivi individuel qui lui permet de se rendre compte de son évolution. L'éducateur peut aussi personnaliser le logiciel en créant ou en supprimant de nouveaux exercices. **EDUCATEMPS** n'est présenté qu'avec quelques exercices dans chaque catégorie, mais ce n'est pas un logiciel figé, donc en créant des exercices avec des niveaux de difficulté différents, les éducateurs peuvent obtenir une grande quantité d'exercices. De plus, nous avons privilégié le thème du temps alors que ce logiciel peut faire travailler les élèves sur d'autres sujets.

Cette évolution est aussi permise grâce au langage de programmation JAVA que nous avons utilisé. Le principe « objet » de ce langage nous fait obtenir une structure du programme relativement claire. Cela pourra permettre à d'autres développeurs d'utiliser notre logiciel comme support et d'ajouter des exercices.

Ce projet a été très intéressant à réaliser car il nous a permis d'étoffer nos connaissances sur le langage JAVA. De plus, c'est motivant de savoir que le logiciel va vraiment être utilisé pour apprendre des notions indispensables à des élèves qui en ont fortement besoin. Nous allons également leur rendre visite à l'IMP de St Sever durant la semaine qui précède la soutenance et nous pourrons vous parler plus en détail des élèves.