Rapport Final — Info 2B

Arthur PIARD

6 Mai 2018

Il existe deux manières de concevoir un programme. La première, c'est de le faire si simple qu'il est évident qu'il ne présente aucun problème. La seconde, c'est de le faire si compliqué qu'il ne présente aucun problème évident. Mais la première méthode est de loin la plus complexe. [0]

0. Auteur inconnu

Première partie

Hiérarchie de classes – Classe abstraite

1. L'héritage entre les classes est l'une des bases de la programmation orientée objet. L'utilité de l'héritage est de pouvoir créer des classes dites filles qui héritent des classes de base dites mères. Un avantage de l'héritage c'est que nous pouvons créer autant de classes héritées de la classe de base que nous voulons. L'héritage va encore plus loin. En effet, il est possible à partir d'une classe fille de créer une nouvelle classe qui hérite de la classe fille.

2. Une classe abstraite est presque comme une classe de base, la grande différence avec une classe de base est qu'on ne peut pas instancier une classe abstraite. Une classe devient abstraite dès lors qu'une méthode abstraite est créée. Dans l'application Harry Potter une classe abstraite est la classe Activite (la classe mère), trois autres classes héritent de la classe mère qui sont Apprentissage, CombatSort, CombatPotion (les classes filles). L'utilité d'avoir la classe mère est qu'on regroupe tout ce qui est commun entre les classes filles. Dans notre cas il s'agit d'une action, la méthode abstraite « execute() ». En effet cette méthode permet la réalisation de l'activité, dans un cas il s'agira d'un apprentissage, dans un autre d'un combat. Lorsque les classes filles héritent de la classe mère, elles héritent aussi de la méthode abstraite « execute() » mais puisqu'elle ne possède pas de corps, les classes filles devront redéfinir cette méthode.

3. (a) Apprentissage :

```
public boolean execute()
{
Personnage oc = getLieu().getOccupant();
// on recupere le PNJ
int n=(int)(Math.random()*200);
// entier n genere pour le booleen gagne
boolean gagne=getJoueur().getExperience()+n>=oc.getExperience();
if (gagne)
// si XP du Joueur + n >= XP du PNJ
{
getJoueur().getArmes().ajoutArme(js);
// le joueur gagne le sort
getJoueur().setExperience(getJoueur().getExperience()+js.getForce());
// le joueur gagne la force du sort en XP
this.setDescriptif("Vous avez appris le sort "+js.getNom());
```

```
// information au Joueur qu'il gagne le sort
}
else
this.setDescriptif("Vous n'avez pas appris le sort "+js.getNom());
// sinon il ne gagne rien
return true;
}
```

(b) CombatPotion :

```
public boolean execute()
{   String result="";       // description de l'activite
int s = 0;                  // somme des forces des potions
boolean res = false;        // savoir si le combat a eu lieu
Potion potion = ((Potion)getLieu()
.getOccupant().getArmes().getPotions()
.getArme((int)(Math.random()*getLieu()
.getOccupant().getArmes()
.getPotions().getNbArme())));
// Potion du PNJ prise aleatoirement
if(this.lp.getNbArme()!=0)
// si le nombre d'arme est different de 0
{
for(Arme ar: lp.getLstArmes().subList(0, lp.getNbArme()-1))
// supprime les doublon des potions
{
s += ((Potion)ar).getForce();
// additionne les forces des potions
}
getJoueur().getArmes().supprArme(utiliser);
// supprime la potion utilisee
boolean gagne = s>=potion.getForce();
if(gagne == true)
// si somme des forces du joueur >= force de la potion du PNJ
{
getJoueur().getArmes().ajoutArme(potion);
// ajout de la potion au joueur
getJoueur().setExperience(getJoueur().getExperience()+potion.getForce());
// ajoute la force de la potion du PNJ en tant qu'XP
result = "Victoire ! Force de vos Potions : "+s+"\n"
+"Force de la potion de l'adversaire : "+potion.getForce()+
"\n Vous gagnez la potion :"+potion+" et "+potion.getForce()+" exp";
// descriptif de l'action
}
else {
getJoueur().setExperience(getJoueur().getExperience()-s);
result = "Defaite ! Force de votre Potion : "+s+
"\n"+"Force de la potion de l'adversaire : "+potion.getForce()+
"\n Vous perdez : "+s+" exp";
}
// descriptif de l'action, il perd la potion et de l'XP
this.setDescriptif(result);
res = true;
}
return res;
}
```

(c) CombatSort :

```
public boolean execute()
{    boolean res = false;       // savoir si le combat a eu lieu
String result="";          // description de l'activite
Sort s = ((Sort)getLieu().getOccupant().getArmes().getSorts()
.getArme((int)(Math.random()*getLieu()
.getOccupant().
getArmes().getSorts().getNbArme())));
// recupere un sort du PNJ aleatoirement
boolean gagne = this.js.getForce()>=s.getForce();
if(js!=null)
// si les armes existent
{
if(gagne == true)
// si la force du sort du Joueur >= force  sort du PNJ
{     getJoueur().getArmes().ajoutArme(s);
// le joueur gagne le sort
getJoueur().setExperience(getJoueur().getExperience()+s.getForce());
// le joueur gagne la force du sort du PNJ en XP
result = "Victoire ! Force de votre sort : "+this.js.getForce()+
"\nForce du sort du combattant : "+s.getForce()+"\n";
result+= "Sort obtenu : "+s+" \nainsi que : "+s.getForce()+" exp";
// descriptif de l'action
}
else
// sinon
{
getJoueur().setExperience(getJoueur().getExperience()-this.js.getForce());
result = "Defaite ! Force de votre sort   : "+this.js.getForce()+
"\nForce du sort du combattant : "+s.getForce()+"\n";
result+= "Vous perdez : "+this.js.getForce()+" exp";
// descriptif de l'action, le joueur perd la force de son sort en XP
}
this.setDescriptif(result);
res = true;
}
return res;
}
```

Deuxième partie

Boîte de dialogue « ApprendreDlg »

1. ApprendreDlg :

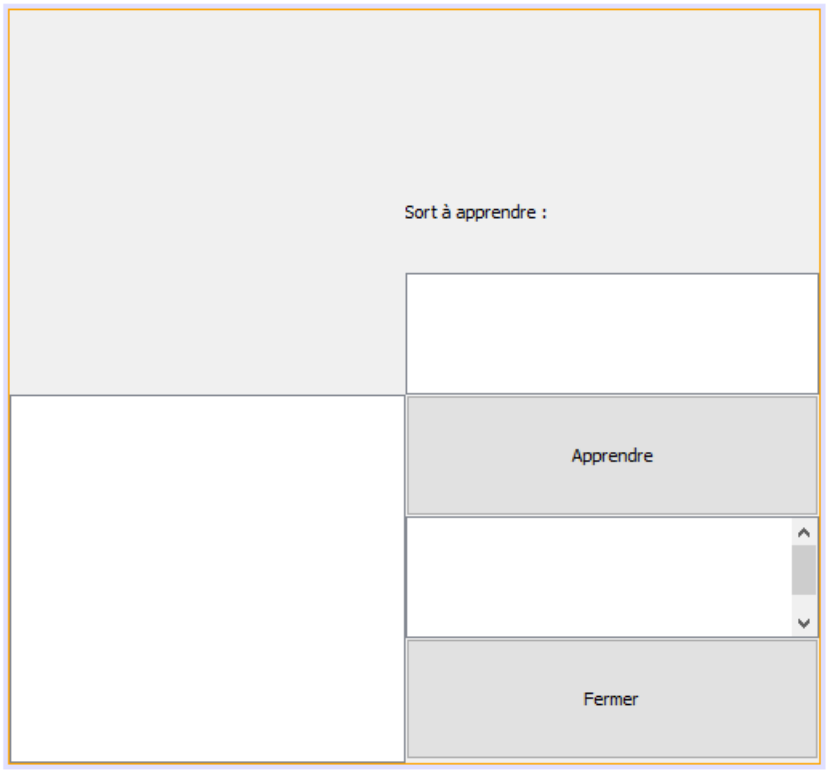

FIGURE 1 – Image de l'interface graphique

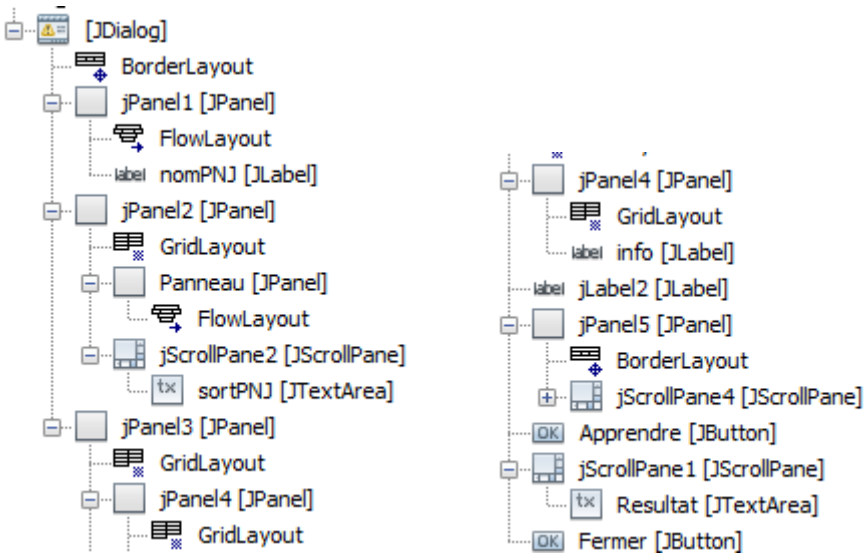

FIGURE 2 – Arborescence complète

2.
```
// Attributs
private Apprentissage act;  // activite du lieu, ici un apprentissage
private boolean ok;
// booleen qui permet de savoir si l'action a eu lieu ou pas
private Lieu lieu;  // Lieu de l'action
private Personnage pc;  // personnage qui fait l'action
private Image img; // Image pour permettre l'affichage
//de la photo du PNJ
private Sort js;  // sort choisi pour l'apprentissage



// Constructeur de la classe
public ApprendreDlg(java.awt.Frame parent, boolean modal,
Personnage p, Lieu l1) {

super(parent, modal);
initComponents();
// creation de la JDialog
initPanneau(img);
// creation du Panneau pour les imgs
pc=p;
// on fixe le personnage qui arrive dans ce lieu a p
lieu=l1;
// on fixe le lieu de l'activite a l1
```

```
sortPNJ.setText(lieu.getOccupant().getArmes().toString());
// on affiche les sorts du PNJ
img=lieu.getOccupant().getPhoto();
repaint();
// on affiche la photo du PNJ
//sur le panneau grace a repaint()
nomPNJ.setText(lieu.getOccupant().getNom());
// on affiche le nom du PNJ
initSorts();
// creation de la liste des sorts a apprendre
this.setSize(530,450);
this.setTitle("Apprendre");
this.setVisible(true);
this.setLocation(600,200);
// on fixe la taille, le titre, la position de la fenetre
}
```

Ne sont utilisés que trois attributs nécessaires à l'activité, un personnage, un lieu et un sort qui sont fixé ou récupérés suivant les besoins. Pour ce qui est du PNJ (Personnage Non Joueur) les informations sont récupérées et affichées sur l'interface de la boite de dialogue.

3. Les informations qui sont transmises à la boite de dialogue sont un personnage ainsi qu'un lieu. La boite de dialogue contient ces informations grâce au constructeur, mais lorsqu'on appuie sur le bouton permettant d'ouvrir cette boite, on crée une instance de ApprendreDlg et on lui donne en paramètre le personnage joueur ainsi que le lieu de l'activité. En effet, on choisit un sort et la méthode execute() se lance. Si l'apprentissage est réussi, il est ajouté à notre collection de sorts, sinon on ne gagne rien, on doit quitter la fenêtre et c'est au tour de l'autre joueur.

4. Après l'action, les informations des joueurs ont été changées, la méthode « afficheJoueur(int i, Joueur p) » permet d'actualiser les informations du joueur passé en paramètre. Les informations sont donc changées une fois l'activité finie et si on gagne un sort, le sort gagné apparaîtra dans l'affichage des sorts du joueur courant.

```
private void ApprendreActionPerformed(java.awt.event.ActionEvent evt) {
ApprendreDlg apprendre = new ApprendreDlg(this, true, pc, ll.getLieu
(CBLieu.getSelectedIndex()-1));
// on creer une instance de ApprendreDlg avec
// en parametre le joueur courrant ainsi que le lieu
//choisi dans le ComboBox
if (apprendre.cOk()) {
// si l'action a ete effectuee alors
changementTour();
// on change de tour
afficheJoueur(1,p1);
```

```
afficheJoueur(2,p2); // on actualise les infos des joueurs
}
}
```

Troisième partie

Gestion de l'affichage des images

1. Un JPanel ImageLieu est créé afin d'afficher l'image du lieu en fonction du lieu choisi dans la JComboBox CBLieu. Dans la méthode initCollection() on créer les liens des photos des PNJ's et des Lieux, par exemple :

   ```
   Image img1 = Toolkit.getDefaultToolkit()
   .getImage(getClass().getResource("Images/foret.jpg"));
   ```

   Ensuite on crée un certain lieu puis on attache la photo à ce lieu par exemple :

   ```
   Lieu4.setPhoto(img10);
   repaint();
   ```

   Noter bien que la méthode repaint() est une surcharge de la méthode paint() qui permet d'afficher la photo dans le JPanel ImageLieu. Cette méthode paint est la suivante :

   ```
   public void paint(Graphics g)
   {
   super.paint(g);
   if(img != null)
     {
           Graphics gg= ImageLieu.getGraphics();
           gg.drawImage(img, 0, 0, ImageLieu.getWidth(), ImageLieu.getHeight(), this);
     }
   }
   ```

   Elle dessine l'image de la photo dans le JPanel ImageLieu avec les dimensions du JPanel, cela donne une image plus propre que de 'simplement' attacher une image à un JButton.

2. Pour l'affichage de la photo du Joueur sélectionné dans la fenêtre ChoixDlg il s'agit quasiment de la même manipulation. Le JPanel s'appelle Photo au lieu de ImageLieu, mais la méthode paint() est la

même en remplaçant ImageLieu par Photo. A titre d'exemple, lorsqu'on clique sur Harry Potter, le lien de l'image de la photo d'Harry Potter est :

```
Image img1 = Toolkit.getDefaultToolkit().getImage(
getClass().getResource("Images/harrypotter.jpg"));
```

Un attribut img de type Image est disponible, à la fin du 'case' 1 dans le switch(int i) , qui est destiné au personnage 1 (Harry Potter) on dessine la photo de la sorte :

```
img=img1;
repaint();
```

L'image est alors dessinée et la méthode est exactement la même pour les autres personnages du Jeu.

3. Pour la boite de dialogue ApprendreDlg la manipulation est presque identique. Un JPanel nommé Panneau existe, simplement on va ajouter à ce JPanel un élément de type PanneauImage, on va forcer le JPanel a passer en GridLayout(1,1), le tout dans une méthode initPanneau(). Le code à l'appui est :

```
private void initPanneau(Image img){
//creation du panneau
PanneauImage pi = new PanneauImage(img);
Panneau.add(pi);
Panneau.setLayout(new GridLayout(1,1));
}
```

La méthode paint() est strictement la même que précédemment, simplement le JPanel s'appel Panneau et non Photo. Dans le constructeur, à l'attribut img de type Image on affecte la photo du PNJ :

```
img=lieu.getOccupant().getPhoto();
repaint();
```

Puis on crée une surcharge de la méthode paint() afin de dessiner la photo du PNJ dans le JPanel.

Quatrième partie

Application principale : HarryPotter

1. Les attributs retenus dans la Classe HarryPotter sont les suivants :

```
private LesLieux ll;
private Personnage lj[];
private Personnage p1;
private Personnage p2;
private LesArmes la = new LesArmes();
private Personnage pc;
private Image img;
private Image couverture = Toolkit.getDefaultToolkit().getImage(
getClass().getResource("Images/couverture.jpg"));
private int MaxExp = 250;
private BoutonImage bImg;
```

   (a) Il y a donc une collection des lieux 'll', qui est utilisée afin d'ajouter les lieux créés un à un, une collection de personnage qui est utilisée elle aussi afin d'ajouter un à un les personnages joueurs du jeu.

   (b) Il y a des personnages p1, p2 qui sont le joueur 1 et le joueur 2 ainsi qu'un personnage pc qui est le personnage courant (c'est à dire celui qui joue).

   (c) Une collection d'armes est présente, dans laquelle sont contenues toutes les armes du jeu (potions et sorts).

   (d) Un entier MaxExp qui fixe le nombre d'expériences à atteindre afin de gagner une partie.

   (e) De plus, un attribut img de type Image est utilisé pour fixer les photos des personnages joueurs et des personnages non joueurs (PNJ). Un autre attribut couverture de type Image est l'image de base sur le JPanel ImageLieu lorsqu'aucun lieu n'est encore choisi.

   (f) Enfin un attribut bImg de type BoutonImage est instancié dans le constructeur avec la surcharge de la méthode paint(). Elle permet de dessiner l'image couverture dans le JPanel ImageLieu au lancement de l'application.

   Les attributs qui sont cités plus haut sont nécessaires car ils sont réutilisés un peu partout dans la classe principale. Si un attribut est retiré, des erreurs apparaîtrons. Le fait que ces attributs soient présents évite la création de variables locales inutiles.

2. Le constructeur de la classe HarryPotter :

```
public HarryPotter()
{
bImg = new BoutonImage(couverture);
repaint();
initComponents();
initArmes();
initPersonnageJoueursTP(la);
initPersonnagesNonJoueursTP(la);
initCollection();
initCombo();
initInfo(false);
Action.setVisible(false);
p1 = null;
p2 = null;
Combattre.setVisible(false);
Apprendre.setVisible(false);
}
```

(a) Comme indiqué plus haut, les deux premières lignes du constructeur permettent d'afficher sur le JPanel ImageLieu l'image de base 'couverture' puisqu'aucun lieu n'est choisi au démarrage de l'application.

(b) La méthode initArmes() comme son nom l'indique crée la collection d'armes.

(c) initComponents() est la méthode qui crée l'interface.

(d) initPersonnageJoueursTP(la) est une méthode qui prend en paramètre la collection d'armes et qui permet de créer les personnages joueurs du Jeu.

(e) initPersonnageNonJoueursTP(la) est une méthode qui prend en paramètre la collection d'armes et qui permet de créer les personnages non joueurs du jeu.

(f) initCollection() est la méthode qui créer la collection des lieux et qui fixe les PNJ dans les lieux en questions.

(g) initCombo() est une méthode qui initialise la ComboBox CBLieu, elle ajoute les descriptions des lieux à la ComboBox.

(h) initInfo(boolean b) est une méthode qui rend visible le JPanel qui affiche les caractéristiques quand le booléen est fixé à true, et qui le cache dans le cas contraire.

(i) le jButton 'Action' n'est pas visible tant que les joueurs ne sont pas ajoutés. Au lancement de l'application puisqu'aucun joueur n'est créé, p1 et p2 sont fixés à null.

(j) De même, les boutons 'Apprendre' et 'Combattre' ne sont pas visibles tant que le jeu n'est pas lancé.

3. le gestionnaire de clic sur le bouton « Apprendre » lance la méthode suivante :

```
private void ApprendreActionPerformed(java.awt.event.ActionEvent evt) {
ApprendreDlg apprendre = new ApprendreDlg(this, true,
pc, ll.getLieu(CBLieu.getSelectedIndex()-1));

if (apprendre.cOk()) // si l'action est effectuee
 {
changementTour();
afficheJoueur(1,p1);
afficheJoueur(2,p2);

 }
}
```

Une instance de ApprendreDlg 'apprendre' est créée avec en paramètre le joueur courant ainsi que l'index du lieu qui a été choisi dans la ComboBox CBLieu. La fenêtre ApprendreDlg s'ouvre alors. Une fois l'action effectuée, on change de tour et on actualise les caractéristiques des deux joueurs. En effet si lors de l'apprentissage le joueur courant a appris un sort, ce sort sera affiché après la fermeture de la boite de dialogue.

Cinquième partie

Boîte de dialogue AjoutLieuDlg (TP8)

On accède à cette fenêtre avec le menu déroulant : 'Gestion' puis 'Ajouter un lieu'. Image de la fenêtre :

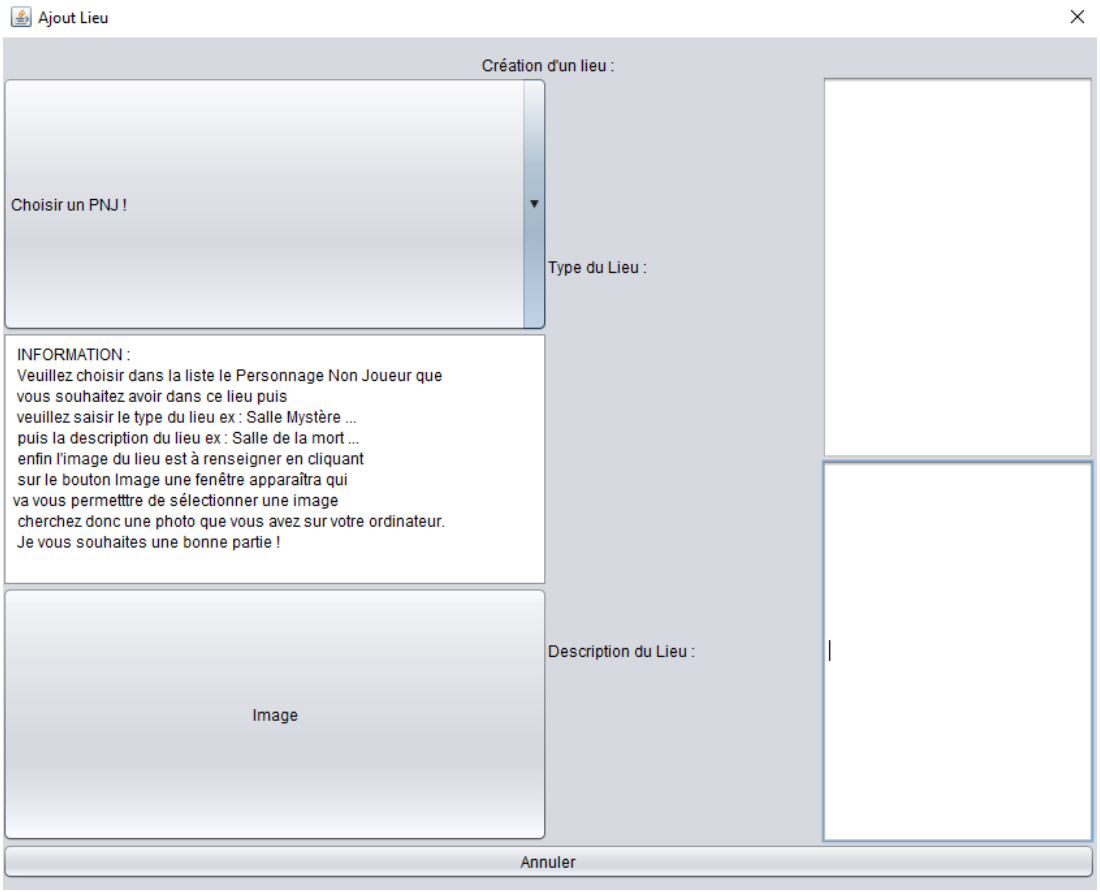

FIGURE 3 – Ajout d'un lieu

1. Les attributs retenus dans la Classe AjoutLieuDlg sont les suivants :

```
private Personnage ljdiag[];
private LesLieux ll;
private Lieu nouveauLieu;
private Image img;
private Personnage occ;
private String path;
```

   Nous avons donc :

   (a) un tableau de Personnage qui contient les PNJ's

   (b) la collection des lieux

   (c) un attribut de type Lieu 'nouveauLieu' qui permet de créer le nouveau lieu à partir de cet attribut

   (d) un attribut de type Image 'img' qui permet de stocker l'image du nouveau lieu

   (e) un attribut de type Personnage qui permet de stocker le PNJ qui occupe ce lieu

   (f) un attribut de type String qui stocke l'adresse de l'image

2. Le constructeur de la classe AjoutLieuDlg :

```
public AjoutLieuDlg(java.awt.Frame parent, boolean modal,
Personnage lj[], LesLieux lesL)
{

super(parent, modal);
initComponents();
ljdiag = lj;
ll = lesL;
information.setText("Note d'information trop longue pour le rapport");
initComboPNJ();
nouveauLieu = null;
this.setSize(800,700);
this.setTitle("Ajout Lieu");
this.setVisible(true);
this.setLocation(600,200);

}
```

   Le tableau de Personnage des PNJ est récupéré ainsi que la collection des lieux. Une note d'information est créée sur la JTextArea 'info'. Une méthode initComboPNJ() permet d'initialiser la ComboBox avec les PNJ's. Enfin puisqu'au lancement de l'application aucun lieu n'a été créé nouveauLieu est initialisé à null. Le reste des commandes rend la fenêtre visible, la place, la dimensionne et lui donne un titre.

Un seul gestionnaire de clic est présent : c'est lorsqu'on clique sur le bouton Image.

```
private void openActionPerformed(java.awt.event.ActionEvent evt)
{
JFileChooser jf = new JFileChooser();
jf.setVisible(true);
if ( jf.showOpenDialog(this) == JFileChooser.APPROVE_OPTION)
// ouvre la JFileChooser
 {

path = jf.getSelectedFile().getPath();
// chemin de l'image
String typeL = typeLieu.getText();
// on recupere le texte ecrit sur le JTextField 'TypeLieu'
String descL =  descLieu.getText();
// on recupere le texte ecrit sur le JTextField 'descLieu'
occ = ll.getLieu(occupantCombo.getSelectedIndex()-1).getOccupant();
// on recupere le PNJ
nouveauLieu = new Lieu(typeL, descL, occ);
// on creer un nouveau lieu avec les parametres
img = Toolkit.getDefaultToolkit().getImage(path);
// on donne le chemin de la photo a img
nouveauLieu.setPhoto(img);
// on fixe la photo au nouveau lieu
ll.ajoutLieu(nouveauLieu);
// on l'ajoute a la collection
this.dispose();
// on ferme la fenetre

 }
}
```

Comme écrit en commentaire dans le code, on récupère la description avec la variable 'descL', le type du Lieu avec la variable 'typeL' ainsi que le PNJ qui occupera le lieu avec la variable 'occ'. On crée le nouveau lieu avec ces trois paramètres puis on lui ajoute une photo avec le chemin qui est dans l'attribut path, enfin on ajoute le lieu à la collection et on ferme la fenêtre.

Gestionnaire de clic AjoutLieuDlg de la classe HarryPotter :

```
private void ajoutLieuActionPerformed(java.awt.event.ActionEvent evt) {

AjoutLieuDlg ajout = new AjoutLieuDlg(this, true, lj, ll);
int i = ll.getNbl();
// on recupere le nombre de lieu
initCombo();
// on rafraichi la combobox CBLieu
img=ll.getLieu(i-1).getPhoto();
// on affiche la photo du lieu
Edition.setText(ll.getLieu(i-1).getDescription());
```

```
// on affiche la description du lieu
}
```

On peut donc voir que lorsque la fenêtre AjoutLieuDlg se ferme, les informations sont récupérées et transmises à l'application principale, une impression d'écran fera l'office d'exemple :

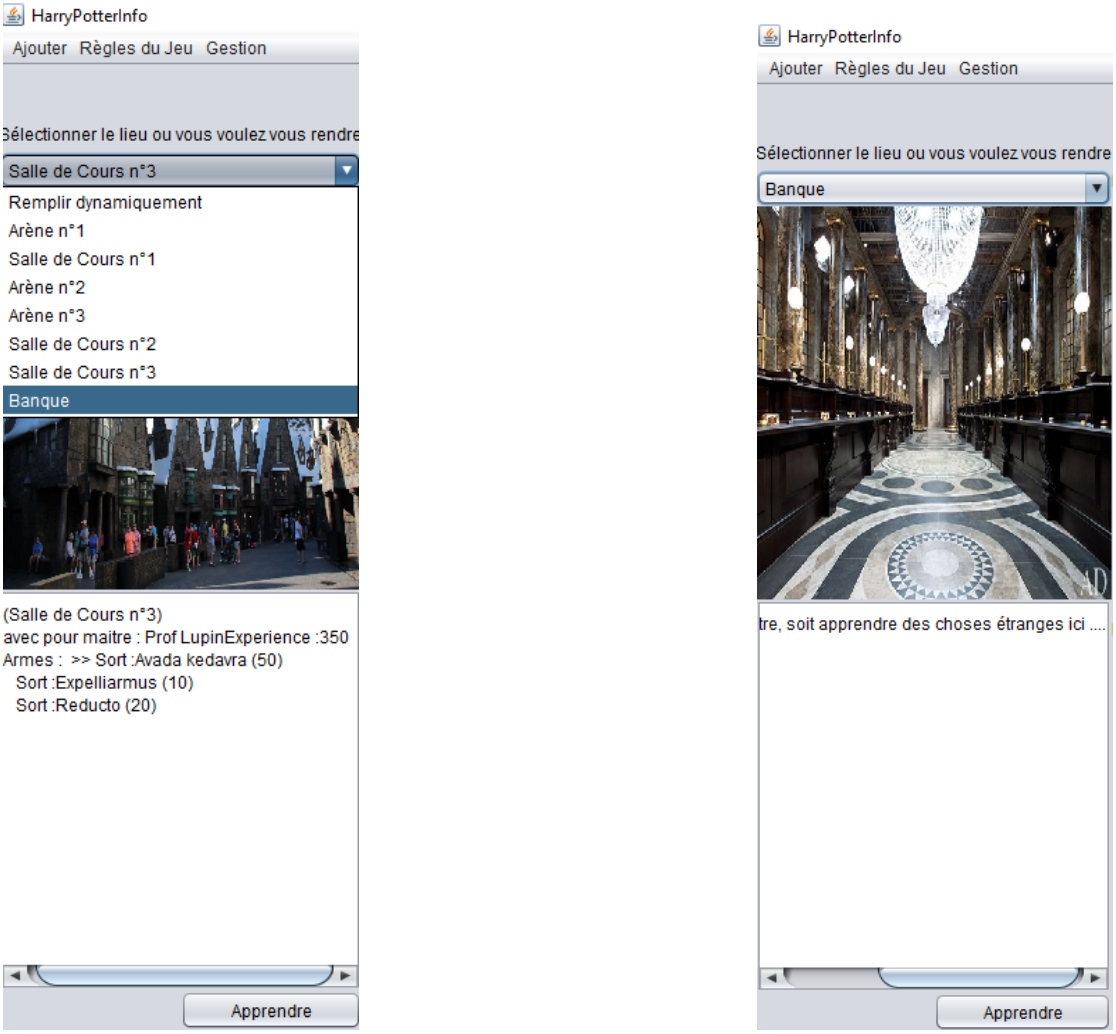

FIGURE 4 – Création du lieu Banque

De plus nous pouvons aussi apprendre ou combattre dans les nouveaux lieux. Dans le lieu 'Banque' de l'exemple le professeur Lupin a été pris comme PNJ. C'est un PNJ qui permet un apprentissage, il est alors possible d'apprendre un sort. De même, si le PNJ choisi est un PNJ qui permet normalement un combat, un combat sera possible. Impression d'écran de démonstration :

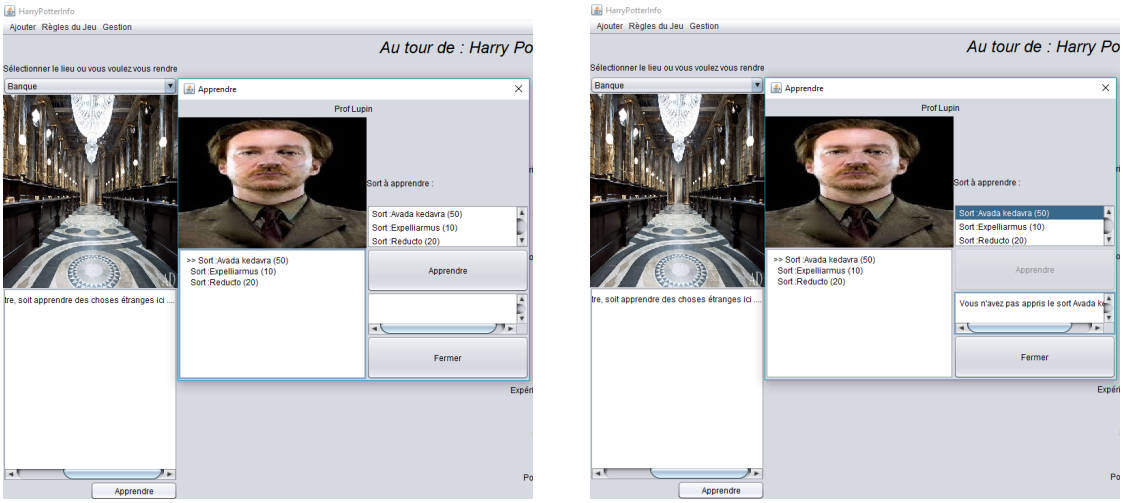

FIGURE 5 – Action dans le lieu Banque

Sixième partie

Conclusion

Le module d'Info 2B a été très intéressant, à mon sens beaucoup plus que l'année dernière. Étant redoublant il me semble que le fait d'être plus encadré au départ sur les TP donne beaucoup plus de motivation. On se sent moins isolé. C'était un projet très intéressant à coder, même si j'ai rencontré des difficultés à certains moments. Je trouve que le point ludique et stimulant de la programmation est de trouver des solutions lorsqu'on est réellement bloqué, soit en corrigeant le problème à la source soit en passant outre en essayant de 'masquer' le problème. J'ai apprécié que nous ayons eu beaucoup de temps pour préparer ce projet. Je vous remercie d'avoir porté attention à mon rapport et j'espère qu'il répondra à vos attentes.

Rapport fait avec LaTeX
par Arthur PIARD