Projet : deuxième partie

Rendre cette partie dans son état d'avancement, quel qu'il soit, le 10 mai 2009 à minuit au plus tard.

April			1	2	3	4	5
	6	7	8	9	10	11	12
	13	14	15	16	17	18	19
	20	21	22	23	24	25	26
	27	28	29	30			
May					1	2	3
	4	5	6	7	8	9	10

L'objectif de cette partie du projet est d'implémenter un module de traitement du français. Ce traitement d'une chaîne de caractères en entrée peut se décomposer en des phases successives :

1. normalisation de l'entrée,
2. découpage en lexèmes,
3. reconnaissance lexicale,
4. analyse syntaxique,
5. résolution des actions.

1 Normalisation

L'objectif de la normalisation est de simplifier la chaîne de caractères d'entrée pour faciliter son traitement. Par exemple, on pourrait vouloir réécrire « au bateau » en « à le bateau », « l'appareil photographique » en « le appareil photographique » etc.

Il faut veiller à faciliter l'ajout de nouvelles normalisations.

2 Découpage en lexèmes

Le découpage de la chaîne d'entrée en mots peut être fait par ANTLR comme vu dans le TP 6. Attention à permettre les caractères accentués, qui pour le français vont de `00C0` à `00FF` plus `0153` en UTF-8.

3 Reconnaissance lexicale

La reconnaissance lexicale va s'appuyer sur un lexique morphologique de grande taille pour le français : Morphalou. Il faut le télécharger depuis l'adresse `http://www.cnrtl.fr/lexiques/morphalou/` puisque sa licence ne permet pas une libre redistribution.

3.1 Morphalou

L'archive au format zip contient un fichier XML de 155 MiB et une DTD. Les entrées de Morphalou sont des *lemmes*, par exemple `championne_1`, qui sont des simplement des identifiants pour des ensembles de *formes* fléchies, par exemple `championne` et `championnes` :

```
<lexicalEntry id="championne_1">
  <feminineVariantOf target="champion_1">champion</feminineVariantOf>
  <formSet>
    <lemmatizedForm>
      <orthography>championne</orthography>
      <grammaticalCategory>commonNoun</grammaticalCategory>
      <grammaticalGender>feminine</grammaticalGender>
    </lemmatizedForm>
    <inflectedForm>
      <orthography>championne</orthography>
      <grammaticalNumber>singular</grammaticalNumber>
    </inflectedForm>
    <inflectedForm>
      <orthography>championnes</orthography>
      <grammaticalNumber>plural</grammaticalNumber>
    </inflectedForm>
  </formSet>
  <originatingEntry target="TLF">CHAMPION, ONNE, subst.</originatingEntry>
</lexicalEntry>
```

Les formes fléchies précisent diverses informations, comme la catégorie syntaxique, le genre, le nombre, etc.

3.2 Chargement de Morphalou

Morphalou pose plusieurs difficultés. La première est sa taille imposante, pour laquelle on conseille

1. de charger le fichier ZIP directement depuis Java, en utilisant `java.util.zip.ZipFile`,
2. de parser son format XML directement en SAX en écrivant votre propre spécialisation de `org.xml.sax.helpers.DefaultHandler`, de manière à économiser en mémoire par rapport à un parser DOM.

Toutes les entrées de Morphalou ne sont pas utiles : seules les adjectifs, noms communs et verbes à l'infinitif ou sous forme de participe passé nous intéressent. En particulier la catégorie des mots fonctionnels de Morphalou n'est pas assez précise pour nos besoins : elle ne distingue pas entre déterminants, pronoms, ou prépositions par exemple. L'archive fournit un début de lexique dans le même format que Morphalou pour ces entrées : `small_lexicon.xml`, que vous pourrez compléter selon vos besoins.

3.3 Utilisation du lexique

Il faut construire une structure permettant, à partir d'une forme fléchie, de retrouver le lemme correspondant. Une `Map` peut suffire à nos besoins, même si une structure d'arbre lexical serait plus efficace.

De manière à limiter l'ambiguïté sur les formes lexicales (par exemple, « aller » peut être le verbe `aller_2`, ou le nom `aller_1` comme dans « un aller simple pour Paris », « marchand » peut être le nom `marchand_1` ou l'adjectif `marchand_2` comme dans « marine marchande »), on peut donner la priorité aux lemmes utilisés dans le scénario de jeu en cours.

4 Analyse syntaxique

L'analyseur syntaxique peut être généré depuis ANTLR. Ce choix nous limite fortement sur la complexité de la syntaxe que nous pouvons permettre, en particulier par son absence d'ambiguïté. Voici néanmoins des exemples de phrases que l'on pourrait vouloir analyser :

```
Aller au phare.
  (Action aller_2 (Instance phare_1))
Faire l'inventaire.
  (Action faire_1 (Instance inventaire_1))
Aller à la forêt.
  (Action aller_2 (Instance forêt_1))
Prendre la pelle.
  (Action prendre_1 (Instance pelle_1))
Décrire.
  (Action décrire_1)
Laisser le petit bout de ficelle emmêlée.
  (Action laisser_1 (Instance ficelle_1 (Modifier bout_1) (Modifier emmêler_1)))
Parler au marchand.
  (Action parler_1 (Instance marchand_1))
Donner la pièce dorée au gros marchand rouge.
  (Action donner_1 (Instance pièce_1  (Modifier doré_1))
                   (Instance marchand_1 (Modifier gros_2) (Modifier rouge_3)))
```

Le résultat de l'analyse syntaxique devrait être un objet de type `Query` sous une forme semblable à celle des exemples ci-dessus, qui sera ensuite résolue en une action du jeu.

On prendra soin d'intercepter les messages d'erreur pour les présenter convenablement au joueur. Même en cas d'erreur, ANTLR permet de construire une requête partielle qui pourra parfois être résolue (demander confirmation au joueur dans ce cas).

5 Résolution des actions

À partir d'une requête comme `(Action aller_2 (Instance phare_1))`, il reste à déterminer l'action et l'instance auxquelles le joueur fait référence.

5.1 Format des scénarios

Le format de description des scénarios doit être modifié pour contenir les informations lexicales. Voici des indications, à vous de modifier la DTD en conséquence :

Instances Par exemple, on pourra trouver

```
<instance id="key" extends="object">
  <lemma id="clef_1"/>
  <lemma id="clé_1"/>
  <modifier role="post"><lemma id="doré_1"/></modifier>
  <reference id="hidden"/>
</instance>
```

qui déclare plusieurs lemmes possibles pour l'instance `key`, et un modificateur (`doré_1`) pour la distinguer d'autres instances qui auraient des lemmes en commun.

Les rôles possibles pour les modificateurs devraient inclure
- `pre` et `post` pour les adjectifs et les participes passés, et
- `quant` pour des constructions comme « un peu de », « un bout de », « un verre de » etc.

L'effet `name` doit être complété pour afficher une description de l'instance à l'aide d'un lemme et de ses modificateurs.

Actions Comme les actions ne sont pas uniques, il est plus confortable de regrouper les lemmes associés dans un lexique à part, mais il faut alors vérifier que toutes les actions ont bien au moins un lemme associé. Par exemple,

```
<lexicon>
  <entry name="describe">
    <lemma id="explorer_1"/>
    <lemma id="parcourir_1"/>
    <lemma id="décrire_1"/>
    <lemma id="regarder_1"/>
    <lemma id="examiner_1"/>
  </entry>
  <entry name="drop">
    <lemma id="laisser_1"/>
    <lemma id="abandonner_1"/>
    <lemma id="enlever_1"/>
  </entry>
  <entry name="go">
    <lemma id="aller_2"/>
    <lemma id="partir_1"/>
```

```
</entry>
...
```

5.2 Résolution

La résolution consiste à déterminer quelle action et quelles instances dans la portée actuelle sont désignées par le joueur. Il faut s'efforcer d'être robuste dans cette résolution, par exemple en d'implémentant des heuristiques pour gérer les arguments manquants sur les actions et les modificateurs superflus sur les instances. En cas de doute sur comment résoudre une requête, il faut demander au joueur de préciser sa demande.