Qui OZ-ce ?

Projet LINFO1104

Gorby Kabasele Ndonda Thomas Wirtgen

Avril 2021

Introduction

Êtes-vous un garçon ou une fille? Avez-vous les cheveux longs ou courts ? Voici tant de questions que l'on peut vous demander pour vous connaître. Il y a un jeu de société bien connu dans lequel vous devez poser des questions pour trouver quelqu'un: le "Qui est-ce?" Vous connaissez surement ce jeu, mais saurez-vous le faire vous-même ? Vous allez en effet réaliser ce jeu durant le projet de cette année.

Objectif

Dans la solution de base, trois étapes sont nécessaires:

1. Lire la base de données;
2. Construire un arbre de décision pour guider la tâche suivante;
3. Poser les questions au joueur humain pour trouver la personne choisie.

Lire la base de données

Vous devez d'abord lire la base de données de personnages depuis un fichier. Puisque vous n'avez pas vu comment manipuler les fichiers ni le texte en Oz, nous vous fournissons le module qui se charge de lire un fichier et de vous le rendre sous une forme de liste de records. La fonction qui fait cela est définie dans le module ProjectLib (voir le code exemple pour savoir comment l'importer). Utilisez le code suivant pour lire la base de données depuis un fichier :

```
ListOfCharacters = {ProjectLib.loadDatabase file "database.txt"}
```

Nous vous proposons une petite base de données contenant les personnages d'Harry Potter. Comme vous pouvez le voir, celle-ci contient une liste de records, il s'agit donc de données standards de Oz. Le label du record est `character`. Le nom du joueur est accessible avec la feature `1`. Les autres champs du record sont les questions avec les réponses. Les noms et questions sont des atomes :

```
[character('Harry Potter'
           'Est-ce que c\'est une fille ?':false
           'A-t-il des cheveux noirs ?':true
           'Porte-t-il des lunettes ?':true
           'A-t-il des cheveux roux ?':false)
character('Ron Weasley'
          'Est-ce que c\'est une fille ?':false
          'A-t-il des cheveux noirs ?':false
          'Porte-t-il des lunettes ?':false
          'A-t-il des cheveux roux ?':true)
character('Hermione Granger'
          'Est-ce que c\'est une fille ?':true
          'A-t-il des cheveux noirs ?':false
          'Porte-t-il des lunettes ?':false
          'A-t-il des cheveux roux ?':false)]
```

Construire un arbre de décision

Votre première tâche est d'écrire la fonction `BuildDecisionTree` qui prend en paramètre la base de données et renvoie un arbre de décision qui sera utilisé durant la deuxième tâche. Cet arbre que nous appellerons `DecisionTree` peut, par exemple, suivre la structure suivante:

```
<DecisionTree>::= leaf(<List[Atom]>) % une liste de nom de personnes
                | question(<Atom> true:<DecisionTree> false:<DecisionTree>)
```

Dans les noeuds internes (la seconde forme), le premier champ est une question à poser au joueur. Sous la feature `true` (resp. `false`), on trouve un plus petit arbre de décision qui contient toutes les personnes qui répondraient `true` (resp. `false`) à la question. Une fois cette arbre de décision créé, nous pouvons le parcourir de la racine jusqu'à atteindre une feuille en posant la question qui se trouve à chaque nœud. Si le joueur répond `true` (resp. `false`), on recommence avec le sous-arbre sous la feature `true` (resp. `false`). Lorsqu'on arrive à une feuille (la première forme), c'est qu'une personne a été trouvée. Le nom de la personne trouvée est l'unique élément de la liste dans le tuple `leaf`. Puisque rien n'interdit à deux personnes différentes d'avoir toutes les mêmes réponses à toutes les questions, il se peut que la liste contienne plusieurs noms de personnes. Il faut alors toutes les proposer au joueur, car plus aucune question ne peut aider à les départager sans améliorer la base de données. Voici une partie d'un exemple d'arbre de décision pour la base de données exemple :

```
question('Est-ce que c\'est une fille ?'
         true: leaf(['Hermione Granger'])
         false: question('Porte-t-il des lunettes ?'
                         true: leaf(['Harry Potter'])
                         false: leaf(['Ron Weasley']))
        )
```

D'autres arbres sont également valides pour la même base de données. Étant donné que le but (implicite) du jeu est de trouver la personne choisie en posant un minimum de questions, il est préférable de produire un arbre ayant une hauteur la plus petit possible. Pour rappel, la hauteur d'un arbre est la distance entre sa racine et sa feuille la plus basse.Un arbre de décision correct est dit optimal s'il n'existe pas d'autre arbre correct dont la hauteur soit strictement plus petite que sa hauteur.Notez qu'il peut exister plusieurs arbres optimaux, si tous ont la même hauteur.

Construire un arbre ayant la hauteur la plus petite.

Construire un arbre de décision optimal n'est pas une tâche facile. Nous n'allons pas exiger que vous le fassiez. Cependant, vous ne pouvez pas construire un arbre de décision stupide (prendre en premier une question qui ne discrimine qu'un seul personnage!).Voici un petit algorithme qui permet de souvent construire des arbres de décisions proches de l'optimalité. Vous devez implémenter un algorithme au moins aussi bon.

Algorithme arbre presque optimal:

Pour chaque question, vous devez compter le nombre de personnes qui répondraient `true` et ceux qui répondraient `false`. Vous allez ensuite sélectionner une des questions pour laquelle les nombres de `true` et de `false` sont les plus proches. La racine de l'arbre utilisera alors cette question. Pour construire le sous-arbre `true`, vous allez prendre le sous-ensemble des personnes qui répondent `true`, et retirer de l'ensemble des questions la question utilisée à la racine. Vous allez ensuite recommencer l'opération avec ces deux sous-ensembles. Vous ferez également de même pour le sous-arbre false. On dit de l'algorithme qu'il est récursif, cela signifie qu'il s'appelle lui-même jusqu'à trouver la bonne réponse. Quand plus aucune question ne peut discriminer une personne dans le sous-ensemble, vous pouvez alors construire une feuille `leaf` avec la liste des personnes restantes. C'est le cas de base de cet algorithme récursif

Démarrage du jeu et choix de la personne

Nous avons conçu une interface utilisateur graphique (GUI pour Graphical User Interface) qui va vous permettre de jouer de manière agréable. Vous pouvez la lancer via la procédure suivante:

```
Options = opts(characters:ListOfCharacters
               builder:TreeBuilder
               driver:GameDriver
               noGUI:false
               autoPlay:ListOfAnswers)

{ProjectLib.play Options}
```

- `ListOfCharacters`: est la liste d'enregistrements contenant les caractéristiques des personnages créée à partir d'un fichier.
- `TreeBuilder`: est une fonction que vous devez définir qui prend un argument une liste de records. Chaque record contient les les caractéristiques d'un personnage. C'est-à-dire, son nom pour la feature `1` et une séquence de question permettant de la discriminer (voir l'exemple du de la base de donnée de la section "Lire la base de données"). Cette fonction retourne l'arbre de décision.
- `GameDriver`: est une fonction que vous devez définir. Elle reçoit en argument l'arbre construit par `TreeBuilder`. Attention cette fonction doit **toujours** renvoyer `unit`. Elle va appeller les procédures de ProjectLib pour diriger le jeu. Pour ce faire, vous avez besoin de deux routines :
  - `{ProjectLib.askQuestion Question}` pose la question `Question` au joueur, et renvoie sa réponse. Par exemple, si l'utilisateur pense à Hermione Granger,

    `{ProjectLib.ask Question 'Est-ce que c\'est une fille ?'}` renverra `true`.
  - `{ProjectLib.found ListOfNames}` doit être appelée lorsque votre programme a trouvé une `leaf`, c'est-à-dire lorsqu'il a deviné la personne. Cette fonction renvoie le nom de la personne s'il est dans la liste ou `false` s'il n'y était pas (ça veut dire que vous vous êtes trompé !)
- `noGUI`: est un booléen précisant si l'interface graphique doit être utilisée (cfr. Version du programme en ligne de commande).

Les options `characters`, `builder`, `noGUI` et `driver` sont des paramètres **obligatoires** à passer à la fonction `ProjectLib.play`. Il est également possible de passer d'autres paramètres au record `Options`. Par exemple, on peut ajouter au record `Options`, le champ `autoPlay:ListOfAnswers`. Il permet de demander à l'application de jouer sans l'intervention humaine. Pour cela, `ListOfAnswers`, la valeur associée au champ `autoPlay`, contient le record `character` du joueur à faire deviner à votre `GameDriver`. Pour obtenir ce record, utiliser le code suivant:

```
ListOfAnswer = {ProjectLib.loadCharacter file PathToAnswerFile}
```

où `PathToAnswerFile` , le chemin vers le fichier, est indiqué sous forme d'un **String** (et donc délimité par des guillemets "). Attention, votre record character doit contenir toutes les questions (et leurs réponses associées) que votre GameDriver pourrait poser. Si la question n'est pas présente, le jeu renverra une erreur.

Les autres champs que l'on peut passer seront présentés plus tard dans ce document.

Voici un exemple d'implémentation de `GameDriver` qui teste avec des `if` imbriqués et représente l'arbre ci-dessus. Votre code ne doit pas supposer un arbre existant, mais bien s'adapter à n'importe quel arbre construit par votre première étape :

```
fun {GameDriver Tree}
   Result
in
   if {ProjectLib.askQuestion 'Est-ce que c\'est une fille ?'} then
      Result = {ProjectLib.found ['Hermione Granger']}
   elseif {ProjectLib.askQuestion 'Porte-t-il des lunettes ?'} then
      Result {ProjectLib.found ['Harry Potter']}
   else
      Result {ProjectLib.found ['Ron Weasley']}
   end

   if Result == false then
      % Arf ! L'algorithme s'est trompé !
      {Browse 'I don't know everything, I just know what I know.'}
   else
      {Browse Result}
   end
   % Toujours renvoyer unit!
   unit
end
```

La procédure `ProjectLib.play` va appeler `GameDriver` autant de fois que le joueur désire jouer. L'arbre de décision calculé lors de la première phase va donc pouvoir être utilisé plusieurs fois !

Version du programme en ligne de commande

Avoir une interface graphique est un plus pour l'intéractivité d'un programme avec l'utilisateur mais pour tester ce programme et le déboguer ce n'est pas ce qu'il y a de plus pratique puisque cela rend l'automatisation de certaines tâches plus compliquée. Nous vous demandons donc de faire en sorte que votre progamme puisse aussi être utilisé sans passer par l'interface. Pour cela, votre programme doit pouvoir être lancé à partir d'une ligne de commande dans le terminal. Il va lire des options qui lui sont passés en ligne de commande et agir en fonction de celles-ci. Pour cela, vous aller devoir compiler votre pogramme avec le compilateur OZ `ozc` de la façon suivante:

```
>ozc -c program.oz -o program.ozf
```

où `program.ozf` est le programme compilé. Une fois ce programme compilé, le programme peut être exécuté avec les options de la façon suivante:

```
>ozengine program.ozf --db <pathToDB> --nogui --ans <pathToAnswer>
```

- `--db <pathToDB>` est l'option donnant le chemin vers le fichier contenant la base de donnée.
- `--nogui` est l'option précisant si il faut lancer le programme avec une interface graphique ou non.
- `--ans` est l'option donnant le chemin vers le fichier contenant les réponses en `autoPlay` (cfr Démarrage du jeu).

Pour vous aider, le programme `Example.oz` vous montre comment lire les options `--db` et `--nogui` passées en ligne de commande. On vous laisse vous charger de faire l'option `--ans`.

```
%% Assigne à Args un record avec les champs 'nogui et 'db'
Args = {Application.getArgs record('nogui'(single type:bool default:false optional:true)
                                   'db'(single type:string default:CWD#"database.txt"))}
...

%%% Assigne la valeur des options
NoGUI = Args.'nogui'
DB = Args.'db'
```

Pour vous aider, nous vous fournissons un Makefile qui compile les fichiers `.oz` et execute le fichier Example.ozf. La commande suivante, vous permet de lancer le Makefile:

```
>make run DB=<pathToDB>
```

Vous pouvez le modifier si cela vous parait nécessaire. **Attention pour ceux qui utilisent Windows, la commande *make* ne fonctionne pas**.Vous devez d'abord installer un environment Unix sur votre PC. Plusieurs alternatives existent mais nous vous suggérons d'utiliser Cygwin (disponible à https://www.cygwin.com/). Assurez vous d'installer la commande *make* lors de l'installation de Cygwin.

Nous nous attendons à ce que votre programme fonctionne en utilisant la commande *ozengine* mentionné plus haut. Dans la version en ligne de commande votre programme prend donc en entrée une pair de fichiers avec la liste des personnages et le personnage recherché et nous nous attendons à ce qu'il affiche dans la console le résultat avec le format suivant:

```
>Name1,Nam2,Name3
```

où chaque élément représente un personnage dont les réponses correspondent à celles du personnage recherché.

Remarque: Si jamais vous avez une erreur dans le terminal disant que les commandes *ozc* ou *ozengine* ne sont pas connues/trouvées, cela signifie probablement que vous n'avez pas ajouté le chemin vers les exécutables Mozart dans la variable d'environment PATH. Cela se fait automatiquement sur MacOS et Linux mais sur Windows vous devez le faire manuellement. Pour plus d'informations, vous pouvez consulter la page https://courses.edx.org/courses/LouvainX/Louv1.01x/1T2014/9426e2d48ae44309841bdcc09e08d8bc/)

Extensions

Nous avons parlé jusqu'ici de la base à implémenter. Maintenant, parlons un peu des extensions possibles. Chaque extension à une difficulté qui lui est associée. Vous devez choisir un sous-ensemble de ces extensions pour un minimum de 4 étoiles. Vous pouvez obtenir plus que 4 étoiles, un bonus pourrait être accordé à ceux qui ont obtenus plus d'étoiles. Mais attention qu'au-delà des 4 étoiles requises, cela ne vous aidera pas à sauver un mauvais projet. Il vaut donc mieux se concentrer sur la base qui est à soumettre avant de travailler sur les extensions.

Extension	Difficulté
Incertitude dans la base de données	*
Questions non binaires (pas seulement true/false)	**
Incertitude du joueur (true/false/I don't know)	***
Gérer les erreurs du joueur	****
Bouton «oups» (revenir à la question précédente)	***
Ajouter un personnage dans la base de données	**
Améliorer la base de données	**

L'implémentation de ces extensions peut mener à des modifications de la structure de l'arbre de décision.

Incertitude dans la base de données

Le record d'une personne peut ne pas contenir les mêmes champs que celui d'une autre personne. Dans le cas d'une question où le joueur répond true, il faudra donc également garder ceux pour qui vous ne connaissez pas la réponse à cette question.

Astuce: la solution idéale ne modifie que l'étape de construction de l'arbre.

Questions non binaires

Cette extension gère les cas où la réponse n'est pastrueou false. Par ex: «La couleur des cheveux est-elle:». Les réponses possibles pouvant être: noir, brun, blond ou roux. **Cette extension doit aussi fonctionner pour la version en ligne de commande**.

Attention: cette extension complique l'implémentation des autres extensions. Réfléchissez donc bien à la manière d'implémenter vos extensions avant de vous lancer sur celle-ci.

Incertitude du joueur

En plus de pouvoir choisir entre true et false, le joueur pourra maintenant répondre «je ne sais pas» (en anglais: I don't know). L'atome `unknown` sera utilisé pour représenter la non-réponse à une question de la part du joueur. Lorsque cela se produit, vous ne pouvez éliminer aucune personne mais vous devez continuer en posant une nouvelle question pour continuer à éliminer des personnes.Pour autoriser cette réponse dans la GUI, vous devez ajouter le champ allowUnknown:true dans le record Options avant de le passer en paramètre ProjectLib.play.

Cette extension doit aussi fonctionner pour la version en ligne de commande. Pour cela, les questions auxquelles le joueur répondrait "je ne sais" ont pour réponse l'atome `unknown`.

Gérer les erreurs du joueur

Jusqu'ici nous avons supposé que le joueur était sûr de ses réponses et ne pouvait pas se tromper. Dans cette extension nous allons tenir compte que quand le joueur répond, il est possible qu'il ait fait une erreur en répondant true alors que la réponse aurait dû être false.

Mais comment savoir si c'est le cas? **Si un {ProjectLib.found} renvoie false, cela voudra dire que c'est le joueur qui s'est trompé et non votre algorithme qui est incorrect**.

Vous n'allez pas vous arrêter à chaque question en pensant qu'il y a une erreur. Cependant vous devez pouvoir prendre en compte une possible erreur du joueur, et recommencer à partir de cette erreur.

Astuce: Vous pouvez considérer que le joueur n'a fait qu'une seule erreur au plus. Il suffit alors de parcourir à nouveau l'arbre de décision pour trouver les personnes qui ont au plus une différence par rapport aux réponses données. Reconstruisez alors un arbre de questions à poser pour seulement ces personnes. Si aucune personne n'a qu'une seule différence, on suppose alors que le joueur a fait deux erreurs et on parcourt à nouveau l'arbre pour trouver les personnes qui ont deux différences, et ainsi de suite.

Cette extension doit aussi fonctionner pour la version en ligne de commande.

Bouton «oups»

Dans l'extension précédente, nous supposons que le joueur a pu faire une erreur. Cependant, le joueur peut se rendre compte qu'il vient de faire une erreur. Dans cette extension nous alors offrir la possibilité au joueur de

corriger la dernière question dans le cas où il voudrait changer sa dernière réponse. Pour afficher ce bouton dans la GUI, vous devez ajouter le champ `oopsButton:true` dans le record Options avant de le passer en paramètre à `ProjectLib.play`.

Ajouter un personnage dans la base de données

Dans la version de base, on suppose que le joueur pense toujours à un personnage qui est bel et bien présent dans la base données. Quid s'il pense à un autre personnage ? Il peut être intéressant, à ce moment-là, de proposer au joueur de l'ajouter dans la base de données, pour que les appels suivants à `GameDriver` puisse trouver ce personnage si le joueur choisit à nouveau celui-ci.

Pour ce faire, on ajoute une nouvelle fonction dans ProjectLib. `{ProjectLib.surrender}` abandonne, et va demander au joueur humain à quel personnage il pensait. Cet appel va renvoyer son nom sous forme d'atome.

Pour pouvoir l'insérer dans la base de données, vous avez besoin aussi des caractéristiques de ce personnage. `{ProjectLib.surrender}` ne vous donne que le nom. Mais vous avez déjà reçu les réponses à toutes les questions que vous avez posées jusque là.

Si vous avez déjà fait l'extension « Gérer l'incertitude dans la base de données », cela peut vous suffir. Vous n'aurez simplement pas la réponse à toutes les questions. Si vous n'avez pas implémenté cette extension, vous devez alors encore poser au joueur toutes les questions qui vous manquent, tout simplement.

Une fois ceci fait, vous pouvez construire un nouveau record pour le personnage, et l'ajouter à `ListOfCharacters`. Vous devez aussi construire un nouvel arbre de décision.

Ici, on ne vous demande pas de reconstruire l'arbre de décision en entier à partir de rien ! On vous demande d'ajouter le nouveau personnage dans l'arbre existant. C'est plus rapide, mais évidemment ça donnera parfois naissance à un arbre qui est plus éloigné de l'optimalité que ce que vous aviez avant. On nomme ce nouvel arbre `NewDecisionTree`. À présent, il ne vous reste plus qu'à indiquer à la `ProjectLib` que vous voulez changer de base de données et d'arbre de décision. Pour cela `GameDriver`, au lieu de renvoyer unit, doit renvoyer `reconfigure(NewOptions)`, où `NewOptions` a évidemment la même structure que le Options transmis à `ProjectLib.play`. Normalement, seule l'option `ListOfCharacters` devrait être modifiée.

Remarque : l'interface graphique vous permettra d'enregistrer dans un fichier la base de données modifiée, pour la réutiliser plus tard !

Astuce : définissez une fonction `MakeGameDriver` qui prend en paramètre la base de données (`ListOfCharacters`) et l'arbre de décision et qui renvoie la fonction GameDriver appropriée.

Cette extension doit aussi fonctionner en ligne de commande. Pour cela vous aller devoir ajouter une nouvelle option `--new` qui précisera à votre progamme où trouver le fichier contenant le nouveau personnage. Comme dans les autres cas, le programme va utiliser le fichier de réponse pour répondre aux questions se trouvant dans l'arbre de décision. Dans le cas, où le personnage n'est pas trouvé et que vous n'avez pas fait l'extension "Gérer l'insertitude dans la base de donnée", le fichier du nouveau personnage sera utilisé pour compléter les réponses. Ce ficher contient un record répresentant le personnage suivant le format du format de réponse. Pour récupérer le personnage, vous pouvez utiliser la fonction `{ProjectLib.loadCharacter file pathToFile}`. Vous devez ensuite passez ce record à `ProjectLib` en utilisant une option `newCharacter: <recordNewCharacter>`

Attention, le fichier des réponses et le fichier contenant le nouveau personnage représente le même personnage mais des parties différentes. Ces fichiers sont incomplets car un fichier seul ne contient pas l'ensemble des pairs question/réponse mais combinés, ils représentent l'entiéreté d'un personnage.

Améliorer la base de données

Cette extension requiert d'avoir fait « Gérer l'incertitude dans la base de données » au préalable. Avec l'extension « Gérer l'incertitude dans la base de données », vous posez parfois des questions pour lesquelles vous ne connaissez pas la réponse pour chaque personnage. Lorsque la partie est finie, vous savez qui est le personnage. En rétrospective, vous pouvez apprendre de nouvelles caractéristiques pour ce personnage, puisque le joueur vous a donné les réponses pour ce personnage. Il peut alors devenir très intéressant d'améliorer la base de données sur base de ces nouvelles informations. Tout comme dans l'extension « Ajouter un personnage dans la base de données », on vous demande ici, non pas de reconstruire un tout nouvel arbre, mais bien de le transformer pour tirer parti des nouvelles informations. Encore une fois, cela pourra donner lieu à un arbre qui n'est pas aussi proche de l'optimalité que celui que vous auriez obtenu en repartant de zéro, mais c'est beaucoup plus rapide. Reportez-vous à la section précédente pour savoir comment reconfigurer `ProjectLib` pour utiliser votre nouvelle base de données. L'astuce s'applique aussi ici.

Dernières précisions

Le projet se déroulera sur quatre semaines, du vendredi 3 Avril au vendredi 30 Avril à 23h00 (heure belge) au plus. Vous devez former des groupes de deux étudiants et soumettre vos travaux sur INGInious. La tâche sera ouverte la dernière semaine du projet. Ne vous y prenez pas au dernier moment car le serveur sera vite surchargé dans les dernières heures avec la limite.

Vous trouverez dans cette archive :

- Ce document que vous êtes en train de lire
- La librairie `ProjectLib.ozf`
- Un exemple de code interragissant avec `ProjectLib.ozf`
- Une base de données qui servira d'entrée à votre programme. Celle-ci contient des personnes et leurs caractéristiques. Votre programme devra déterminer quelles questions poser, et dans quel ordre, afin de trouver qui est la personne que le joueur a choisie préalablement.

Vous devez tous commencer par faire une solution de base que vous soumettrez dès que celles-ci répondra aux consignes. Ensuite, vous pourrez agrémenter votre solution en implémentant des extensions supplémentaires en bonus.

Votre solution doit être entièrement déclarative. Le partage entre groupe est évidemment interdit et considéré comme du plagiat. Vous pouvez cependant discuter de vos stratégies pour réaliser votre solution mais sans échanger de code.

Dans le rapport, de maximum 5 pages, vous devez expliquer la structure de votre programme, les décisions de conception que vous avez prises et bien sûr la liste des extensions supplémentaires que vous avez implémentées. Mentionnez également les limitations et les problèmes connus de votre programme.

Votre rapport doit également comprendre une analyse de la complexité de `GameDriver`, dans le cas où on reste dans la version de base sans extension supplémentaire (pour plus de simplicité). Pour pouvoir faire cela, vous devez connaître la complexité des fonctions que vous appelez. Nous vous précisons donc que `ProjectLib.askQuestion`, `ProjectLib.found`, `ProjectLib.surrender` peuvent être considérées comme étant $\Theta(1)$.