Université catholique de Louvain

[LINFO1104]
Projet : Qui OZ-ce ?

Simon Cornelis
simon.cornelis@student.uclouvain.be

Vincent Buccilli
vincent.buccilli@student.uclouvain.be

May 7, 2021

1 Fonctionnement du code

Pour ce projet nous avions pour objectif de recréer le jeu « Qui est-ce » en *Oz*. Afin de nous aider, nous avons eu accès à quelques fichiers d'exemple ainsi qu'une librairie *ProjectLib* comprenant un ensemble de fonctions utiles au bon déroulement du jeu ainsi qu'une interface graphique permettant un plus grand confort pour le joueur. Cependant, il nous était malgré tout demandé de faire fonctionner le programme en ligne de commande pour permettre l'automatisation des tests.

Notre approche a donc été de partir de ces exemples, de créer un code fonctionnel et ensuite de l'optimiser et d'implémenter toutes les fonctionnalités demandées ainsi que les extensions. Pour la structure de notre code, nous avons favorisé la création de sous-fonctions simples et courtes, plutôt que peu de fonctions « all-in-one ». Ainsi nous avons les fonctions principales suivantes, dans le fichier *Main.oz*, composées de sous-fonctions dans leurs fermetures :

- *NextQuestion* : retourne, parmi les données restantes, la meilleure question à poser pour diviser le groupe en deux parts les plus égales possibles.
- *TreeBuilder* : fonction produisant un arbre assez équilibré (probablement pas optimal dans tous les cas) de questions à poser au joueur.
- *GameDriver* : fonction parcourant l'arbre passé en argument de façon récursive en posant des questions au joueur pour le choix du sous-arbre à suivre.

1.1 TreeBuilder

Commençons par explorer la fonction *TreeBuilder*. Cette fonction se charge de produire un arbre respectant la définition ci-dessous (structure hybride entre un arbre et une liste). Avec Q qui est la question à poser au joueur et **true** et **false** qui sont les deux chemins possibles, en fonction de la réponse qu'il choisit. Nous avons décidé d'utiliser des listes (possiblement vides) comme feuilles par manque d'intérêt de les emballer dans des record leaf.

```
<Tree T> ::= <List L> % list of answers (nil if none)
           | question(Q true:<Tree T> false:<Tree T>) % Q is an atom
```

Le *TreeBuilder* dépend de la fonction *NextQuestion*. Le cas de base de la récursion, qui interrompt la récursion, est lorsque l'appel à *NextQuestion* renvoie nil. Dans ce cas il n'y a plus de questions à poser au joueur et on peut se charger de transformer la liste de records de personnages en une liste de noms de personnages (atomes). Nous faisons cela (ligne 42) à l'aide d'un *Map* qui applique une fonction sélectionnant la valeur du premier champ de chaque record (il s'agit du nom du personnage), sur chaque record de la liste.

```
40 fun {TreeBuilder Data}
41    case {NextQuestion Data}
42       of nil then {Map Data fun {$ E} E.1 end} % keep only names list
43       [] NextQ then T F ClearQ in
44          fun {ClearQ P} {Record.subtract P NextQ} end % remove question
45          {List.partition Data fun {$ E} E.NextQ end T F} % split T and F
46          question(NextQ
47                   true:{TreeBuilder {Map T ClearQ}}
48                   false:{TreeBuilder {Map F ClearQ}})
49    end
50 end
```

Le second cas possible, serait que *NextQuestion* renvoie un atome question (ligne 43). Ce cas se charge de la récursion. Nous avons défini (ligne 44) une fonction *ClearQ* qui permet de retirer la question NextQ d'un record. On sépare (ligne 45) la liste des personnages en fonction de leur réponse à la question dans deux listes T et F. Et on renvoie un arbre (lignes 46-48) contenant deux sous-arbres, qui sont générés par des appels récursifs, desquels on retire la question qu'on vient d'utiliser.

Il est intéressant de se pencher sur la structure de l'arbre pour se rendre compte que pour l'optimiser et éviter de poser des questions inutiles au joueur, on ne devrait jamais tomber sur une liste vide où que ce soit dans l'arbre. Si la liste est vide, c'est que la question de l'arbre parent n'avait pas besoin d'être posée.

1.2 NextQuestion

L'objectif de *NextQuestion* est de retourner la prochaine question à poser au joueur de façon à minimiser le nombre de questions à poser. L'approche que nous avons choisie est de trouver la question qui divise le groupe de personnages en deux parts les plus égales possibles. L'idée pour compter ça, est de garder des scores pour chaque question. Si un personnage répond **true**, on augmente le score, sinon on le diminue. La question idéale obtient donc un score de 0.

On a pour commencer un record score (ligne 23) contenant les questions restantes avec pour chacune une valeur de 0. On génère ce record en prenant la première personne de Data et en créant une liste de ses features, sans oublier de retirer la première qui est celle contenant le nom du personnage. On crée pour chaque question un tuple pour finalement utiliser la fonction *ToRecord* pour transformer tout ça en un record de type score(q1:0 q2:0 ... qn:0) avec q1, q2, ... qui sont les questions.

```
23 Start={List.toRecord score {Map {Arity Data.1}.2 fun {$ E} E#0 end}}
```

La fonction *ScoreQuestions* est utilisée pour mettre a jour le score actuel (dans un accumulateur Acc) en ajoutant 1 pour chaque question où la réponse est **true** et en retirant 1 sinon, pour le personnage Per actuel. Il va falloir appeler cette fonction sur chaque personnage de la liste afin d'avoir les scores au complet.

```
26 fun {ScoreQuestions Acc Per}
27    {Record.mapInd Acc fun {$ Q S} if Per.Q then S+1 else S-1 end end}
28 end
```

Pour garder uniquement la meilleure question à poser, on réduit la liste avec une fonction qui recherche la question de score minimal, qui est la plus optimale à poser. Elle compare la valeur absolue du meilleur score jusqu'à présent avec le score X de la question Q actuelle. Seulement si il est meilleur, on le prend comme nouveau minimum. Cela a pour effet que l'élément de départ nil#{Length Data} restera tant qu'il n'y a pas de question qui divise mieux le groupe. Par conséquent, si toutes les personnes restantes répondent toujours la même chose, toutes les questions ont un score de {Length Data} ce qui fait que l'élément de départ restera. C'est alors nil qui est renvoyé pour qu'aucune question supplémentaire ne soit posée. Donc si il ne reste qu'une personne, les questions ne seront pas posées (score de 1 ou $\sim$ 1).

```
31 fun {Best Q Acc X}
32    if Acc.2 > {Abs X} then Q#{Abs X} else Acc end
33 end
```

Il ne nous reste plus qu'à calculer les Scores en faisant une réduction des données Data à l'aide de la fonction *ScoreQuestions* sur l'élément de départ Start (contenant des scores nuls). On obtient alors un record où les scores de chaque question ont été calculés. Cela ressemble à score(q1:$\sim$2 q2:5 ... qn:0) avec q1, q2, ... qui sont les questions.

```
35 Scores = {FoldL Data ScoreQuestions Start}
```

La valeur retournée par la fonction *NextQuestion* est calculée avec l'appel ci-dessous. On réduit le record des Scores avec notre fonction *Best* qui va garder uniquement la question à poser ou bien l'élément de départ. L'appel à *Record.foldLInd* permet d'appliquer une fonction qui reçoit l'accumulateur, la valeur actuelle, mais aussi la clé actuelle (qui est la question) sur chaque élément pour réduire la liste. Le retour est de forme identique à l'élément de départ, c'est à dire Question#Score et c'est pour cela qu'on fait un ".1" après pour ne prendre que la question dans le tuple.

```
37 {Record.foldLInd Scores Best nil#{Length Data}}.1
```

1.3 GameDriver

Le *GameDriver* se charge de l'interaction avec le joueur et du déroulement de la partie. Nous avons défini la fonction *Ask* qui se charge de récursivement parcourir l'arbre en posant les questions au joueur, jusqu'à arriver à une réponse où jusqu'à devoir abandonner.

```
53 fun {Ask Tree}
54    case Tree
55       of nil then {ProjectLib.surrender}
56       [] P|Ps then {ProjectLib.found P|Ps} % list of answers
57       [] question(Q true:T false:F) then
58          if {ProjectLib.askQuestion Q} then {Ask T} else {Ask F} end
59    end
60 end
```

Ask traite trois situations possibles dans un **case**. Le cas de base (ligne 55), si on tombe sur une liste vide, c'est qu'on n'a pas de solution pour cet ensemble de réponses que le joueur nous a donné. On doit donc se rendre à l'aide de *ProjectLib.surrender* et cela arrête la récursion. Cependant, un arbre bien construit ne devrait jamais contenir de liste vide donc il s'agit d'une sécurité.

Un second cas de base (ligne 56), est lorsqu'on tombe sur une liste dans l'arbre. Il s'agit alors d'une liste de personnages possibles et on peut les proposer au joueur à l'aide de la fonction *ProjectLib.found*.

Le dernier cas est le plus courant, lorsqu'on obtient un arbre sous forme de question. On pose alors la question Q au joueur avec *ProjectLib.askQuestion*, si il répond **true**, alors on continue récursivement dans le sous-arbre T, sinon dans le sous-arbre F. Ces sous arbres ne contiennent que des personnages qui ont des réponses identiques, pour l'ensemble des questions auxquelles le joueur a déjà répondu.

Le *GameDriver* fait un seul appel à *Ask*. Cette dernière se rapelle récursivement jusqu'au résultat de la partie. On pourrait avoir **false** (ligne 63) si on a proposé une liste mais que le joueur n'a pas trouvé son personnage dans la liste et a appuyé sur le bouton « Mon personnage n'est pas dans la liste ». Sinon il y a deux cas, soit c'est un seul personnage (ligne 65) et on n'a qu'à imprimer son nom. Si c'est une liste (ligne 64), il faut la convertir pour avoir les noms séparés par des virgules. On utilise une fonction réductrice *FoldL* sur la liste en concaténant, sous forme d'un virtual string.

```
62 case {Ask Tree}
63    of false then {FPrint {ProjectLib.surrender}}
64    [] H|T then {FPrint {FoldL T fun {$ A B} A#','#B end H}}
65    [] Result then {FPrint Result}
66 end
```

1.4 Version en ligne de commande

Pour la version en ligne de commande, il faut se charger de la sortie du résultat dans le terminal. Pour cela, nous nous sommes basés sur le fichier *write_example.oz*. Notre but est d'écrire dans le *standard output* donc on commence par ouvrir le fichier *stdout* sous le nom de File et à la fin, juste avant l'appel à *Application.exit*, on le referme. Nous avons également défini une fonction qui permet d'imprimer des virtual strings pour n'avoir qu'à faire un appel à *FPrint* lorsqu'on veut imprimer dans la console. A là fin du *GameDriver*, on imprime un retour à la ligne lorsqu'on n'est pas en mode NoGUI pour avoir une séparation entre les réponses des différentes parties.

```
9 FPrint = proc {$ S} {File write(vs:S)} end
```

2 Extensions

Nous avons implémenté les deux extensions dans le fichier *Extensions.oz* et avons fourni en annexe une comparaison des changements effectués entre la version principale et celle avec les extensions. On voit

ainsi sur la figure 1, en rouge, ce qui a été retiré du fichier *Main.oz* et remplacé par ce qui est, en vert, dans le fichier *Extensions.oz*.

2.1 Incertitude du joueur (extension ***)

L'objectif de cette première extension est de permettre à l'utilisateur d'appuyer sur le bouton « Je ne sais pas » pour répondre par unknown lorsque qu'il ne connaît pas la réponse à une question. Pour cela, nous avons modifié la structure de l'arbre afin d'avoir un chemin unknown. Le sous-arbre unknown ne réduit pas la taille de la liste de personnages possibles. Le *TreeBuilder* se contente de retirer la question qui vient d'être posée de chaque record du sous-arbre. Cela modifie donc la définition de notre structure tel que ceci :

```
<Tree T> ::= <List L> % list of answers (nil if none)
           | question(Q true:<Tree T> false:<Tree T> unknown:<Tree T>)
```

2.2 Bouton « oups » (extension ***)

L'objectif de cette deuxième extension est de permettre une gestion des erreurs par le joueur. Il a désormais la possibilité de revenir en arrière jusqu'à la première question pour annuler ses réponses en appuyant sur le bouton « Oops! ». Cependant, appuyer sur le bouton alors qu'on est revenu jusqu'à la première question n'a plus d'effet, on reste sur place.

L'astuce pour pouvoir revenir en arrière est de garder un historique (comme dans un navigateur internet) sous la forme d'une pile. En sachant que l'implémentation d'une liste en *Oz* ressemble à une liste chaînée, insérer et extraire le premier élément se comporte comme une pile. Nous avons ajouté un second argument Last à la fonction *Ask* du *GameDriver* qui est une liste d'historique d'arbres. A chaque appel récursif de *Ask*, nous ajoutons l'arbre actuel en première position dans la liste (lignes 62-64). Voici le **case** qui remplace le **if** de la ligne 58 de la version sans extensions.

```
59 case {ProjectLib.askQuestion Q}
60    of oops then
61       case Last of H|T then {Ask H T} else {Ask Tree Last} end
62    [] true then {Ask T Tree|Last}
63    [] false then {Ask F Tree|Last}
64    [] unknown then {Ask U Tree|Last}
65 end
```

Lorsque le joueur appuie sur le bouton, l'atome oops est renvoyé. Nous avons donc ajouté un pattern pour cela (ligne 60) et analysons l'historique Last. Soit il s'agit d'une liste, alors on peut appeler *Ask* avec le premier élément de Last et l'historique est la queue de la liste. Sinon la liste est vide, alors c'est qu'il n'y a plus de questions avant et on reste donc sur place.

3 Problèmes et limitations connues

L'ensemble des fonctionnalités demandées dans l'énoncé sont implémentées et fonctionnelles. A la fois la version en ligne de commande et avec interface graphique fonctionnent tel que nous avons compris les consignes.

Les deux extensions imposent des sacrifices en optimisation. L'arbre contenant le chemin unknown est de hauteur maximale. Il s'agit d'un pire cas, si le joueur ne fait que répondre « Je ne sais pas », on est obligés de lui poser toutes les questions. Le fait d'avoir un historique des arbres n'est pas non plus optimal en terme de mémoire, surtout dans le cas où le joueur ne fait que répondre « Je ne sais pas ». On se retrouve alors avec une liste d'arbres qui, en fonction de leurs tailles, peut devenir assez conséquent en mémoire. Cependant, nous avons implémenté ce code pour qu'il soit robuste pour des utilisations normales du jeu, avec un nombre raisonnable de personnages.

Annexes

A Changements pour les extensions

```
40  40      fun {TreeBuilder Data}
41  41        case {NextQuestion Data}
42  42          of nil then {Map Data fun {$ E} E.1 end} % keep only names list or nil (if none)
43  43          [] NextQ then T F ClearQ in
44  44            fun {ClearQ P} {Record.subtract P NextQ} end % remove question from db records
45  45            {List.partition Data fun {$ E} E.NextQ end T F} % split in T and F (answer to NextQ)
46  46            question(NextQ
47  47                     true:{TreeBuilder {Map T ClearQ}}
48      -                  false:{TreeBuilder {Map F ClearQ}})
        48+                false:{TreeBuilder {Map F ClearQ}}
        49+                unknown:{TreeBuilder {Map Data ClearQ}})
49  50        end
50  51      end
51  52
52  53      fun {GameDriver Tree}
53      -     fun {Ask Tree}
        54+   fun {Ask Tree Last} % Last is a list of previous Trees
54  55          case Tree
55  56            of nil then {ProjectLib.surrender}
56  57            [] P|Ps then {ProjectLib.found P|Ps} % list of answers
57      -         [] question(Q true:T false:F) then
58      -           if {ProjectLib.askQuestion Q} then {Ask T} else {Ask F} end
        58+         [] question(Q true:T false:F unknown:U) then
        59+           case {ProjectLib.askQuestion Q}
        60+             of oops then
        61+               case Last of H|T then {Ask H T} else {Ask Tree Tree} end
        62+             [] true then {Ask T Tree|Last}
        63+             [] false then {Ask F Tree|Last}
        64+             [] unknown then {Ask U Tree|Last}
        65+           end
59  66          end
60  67        end
61  68      in
62      -     case {Ask Tree}
        69+   case {Ask Tree nil}
63  70        of false then {FPrint {ProjectLib.surrender}}
64  71        [] H|T then {FPrint {FoldL T fun {$ A B} A#','#B end H}}
65  72        [] Result then {FPrint Result}
66  73        end
```

Figure 1: Différences entre le fichier *Main.oz* et le fichier *Extensions.oz*.

B Fichier *Main.oz*

```
1  functor
2  import
3    Application Browser Open OS ProjectLib System
4  define
5    CWD = {Atom.toString {OS.getCWD}}#"/"
6    Browse = proc {$ Buf} {Browser.browse Buf} end
7    Print = proc {$ S} {System.print S} end
8    File = {New Open.file init(name:'stdout' flags: [write create
       truncate text])}
9    FPrint = proc {$ S} {File write(vs:S)} end
10   Args = {Application.getArgs record(
11            'nogui'(single type:bool default:false optional:true)
12            'db'(single type:string default:CWD#"database.txt")
```

```
13            'ans'(single type:string default:CWD#"test_answers.txt"
                 optional:true))}
14  in
15    local
16      NoGUI = Args.'nogui'
17      ListOfCharacters = {ProjectLib.loadDatabase file Args.'db'}
18      ListOfAnswers = {ProjectLib.loadCharacter file Args.'ans'}
19
20      % get next best question to ask to split possible anwers equally
21      fun {NextQuestion Data}
22        % returns score(q1:0 q2:0 ... qn:0) (each question starts with
             a score of 0)
23        Start = {List.toRecord score {Map {Arity Data.1}.2 fun {$ E} E
             #0 end}}
24
25        % returns a record with the score set for each question like
             score(q1:~2 q2:5 q3:0)
26        fun {ScoreQuestions Acc Per}
27          {Record.mapInd Acc fun {$ Q S} if Per.Q then S+1 else S-1 end
               end}
28        end
29
30        % returns the question with the best score
31        fun {Best Q Acc X}
32          if Acc.2 > {Abs X} then Q#{Abs X} else Acc end
33        end
34
35        Scores = {FoldL Data ScoreQuestions Start}
36      in
37        {Record.foldLInd Scores Best nil#{Length Data}}.1 % Acc =
             Question#Score
38      end
39
40      fun {TreeBuilder Data}
41        case {NextQuestion Data}
42          of nil then {Map Data fun {$ E} E.1 end} % keep only names
               list or nil (if none)
43          [] NextQ then T F ClearQ in
44            fun {ClearQ P} {Record.subtract P NextQ} end % remove
                 question from db records
45            {List.partition Data fun {$ E} E.NextQ end T F} % split in
                 T and F (answer to NextQ)
46            question(NextQ
47                     true:{TreeBuilder {Map T ClearQ}}
48                     false:{TreeBuilder {Map F ClearQ}})
49        end
50      end
51
52      fun {GameDriver Tree}
53        fun {Ask Tree}
54          case Tree
55            of nil then {ProjectLib.surrender}
56            [] P|Ps then {ProjectLib.found P|Ps} % list of answers
```

```
57              [] question(Q true:T false:F) then
58                if {ProjectLib.askQuestion Q} then {Ask T} else {Ask F}
                     end
59            end
60          end
61       in
62          case {Ask Tree}
63             of false then {FPrint {ProjectLib.surrender}}
64             [] H|T then {FPrint {FoldL T fun {$ A B} A#','#B end H}}
65             [] Result then {FPrint Result}
66          end
67
68          if NoGUI == false then {FPrint '\n'} end
69
70          unit % always return unit
71       end
72    in
73       {ProjectLib.play opts(characters:ListOfCharacters autoPlay:
             ListOfAnswers
74                             noGUI:NoGUI builder:TreeBuilder driver:
                                 GameDriver)}
75       {File close}
76       {Application.exit 0}
77    end
78 end
```

C Fichier *Extensions.oz*

```
1  functor
2  import
3     Application Browser Open OS ProjectLib System
4  define
5    CWD = {Atom.toString {OS.getCWD}}#"/"
6     Browse = proc {$ Buf} {Browser.browse Buf} end
7     Print = proc {$ S} {System.print S} end
8     File = {New Open.file init(name:'stdout' flags: [write create
         truncate text])}
9     FPrint = proc {$ S} {File write(vs:S)} end
10    Args = {Application.getArgs record(
11             'nogui'(single type:bool default:false optional:true)
12             'db'(single type:string default:CWD#"database.txt")
13             'ans'(single type:string default:CWD#"test_answers.txt"
                  optional:true))}
14 in
15    local
16       NoGUI = Args.'nogui'
17       ListOfCharacters = {ProjectLib.loadDatabase file Args.'db'}
18       ListOfAnswers = {ProjectLib.loadCharacter file Args.'ans'}
19
20       % get next best question to ask to split possible anwers equally
21       fun {NextQuestion Data}
22          % returns score(q1:0 q2:0 ... qn:0) (each question starts with
               a score of 0)
```

```
23        Start = {List.toRecord score {Map {Arity Data.1}.2 fun {$ E} E
              #0 end}}
24
25        % returns a record with the score set for each question like
              score(q1:~2 q2:5 q3:0)
26        fun {ScoreQuestions Acc Per}
27          {Record.mapInd Acc fun {$ Q S} if Per.Q then S+1 else S-1 end
                end}
28        end
29
30        % returns the question with the best score
31        fun {Best Q Acc X}
32          if Acc.2 > {Abs X} then Q#{Abs X} else Acc end
33        end
34
35        Scores = {FoldL Data ScoreQuestions Start}
36      in
37        {Record.foldLInd Scores Best nil#{Length Data}}.1 % Acc =
              Question#Score
38      end
39
40      fun {TreeBuilder Data}
41        case {NextQuestion Data}
42          of nil then {Map Data fun {$ E} E.1 end} % keep only names
                list or nil (if none)
43          [] NextQ then T F ClearQ in
44            fun {ClearQ P} {Record.subtract P NextQ} end % remove
                  question from db records
45            {List.partition Data fun {$ E} E.NextQ end T F} % split in
                  T and F (answer to NextQ)
46            question(NextQ
47                     true:{TreeBuilder {Map T ClearQ}}
48                     false:{TreeBuilder {Map F ClearQ}}
49                     unknown:{TreeBuilder {Map Data ClearQ}})
50        end
51      end
52
53      fun {GameDriver Tree}
54        fun {Ask Tree Last} % Last is a list of previous Trees
55          case Tree
56            of nil then {ProjectLib.surrender}
57            [] P|Ps then {ProjectLib.found P|Ps} % list of answers
58            [] question(Q true:T false:F unknown:U) then
59              case {ProjectLib.askQuestion Q}
60                of oops then
61                  case Last of H|T then {Ask H T} else {Ask Tree Last}
                        end
62                [] true then {Ask T Tree|Last}
63                [] false then {Ask F Tree|Last}
64                [] unknown then {Ask U Tree|Last}
65              end
66          end
67        end
```

```
68      in
69         case {Ask Tree nil}
70            of false then {FPrint {ProjectLib.surrender}}
71            [] H|T then {FPrint {FoldL T fun {$ A B} A#','#B end H}}
72            [] Result then {FPrint Result}
73         end
74
75         if NoGUI == false then {FPrint '\n'} end
76
77         unit % always return unit
78      end
79   in
80      {ProjectLib.play opts(characters:ListOfCharacters autoPlay:
           ListOfAnswers
81                            noGUI:NoGUI builder:TreeBuilder driver:
                                 GameDriver
82                            oopsButton:true allowUnknown:true)}
83      {File close}
84      {Application.exit 0}
85   end
86 end
```