Projet de programmation C : le devin

L'objectif de ce projet est de développer une application graphique d'un jeu où un personnage de votre choix devine l'objet des pensées de l'utilisateur à l'aide d'une série de questions. Il est ainsi demandé de développer une sorte d'intelligence artificielle pour le jeu pour enfants «Qui est-ce ?». Mais au lieu de travailler avec un plateau fini comportant des images de personnages, votre programme devra manipuler des données de personnages et des questions dynamiques et évolutives.

Description et déroulement du programme

L'application attendue s'inspire largement d'une part du jeu pour enfants «Qui est-ce ??» :

http://fr.wikipedia.org/wiki/Qui_est-ce_%3F

et s'inspire d'autre part de l'application Web Akinator :

http://fr.akinator.com/

Une fois lancé, le jeu se déroule comme il suit :

- Le jeu demande à l'utilisateur de penser très fort à un personnage.
- Le jeu pose une série de 20 questions à l'utilisateur qui doit fournir des réponses en cohérence avec son personnage choisi.
- Le jeu donne sa meilleure proposition correspondant aux réponses données par l'utilisateur.
- L'utilisateur informe le programme si le personnage proposé est le bon.
- Si le programme trouve juste : les données du personnage concerné sont mises à jour. Potentiellement, le programme apprend de nouvelles réponses.
- Si le programme se trompe : on prévoit alors un mécanisme pour rajouter le personnage si ce dernier n'existait pas dans les données du programme et on propose à l'utilisateur de rajouter une question qui aurait permis d'identifier son personnage.

Par souci d'ergonomie, on proposera une interface graphique faisant apparaître les questions au fur et à mesure du déroulement du jeu et permettant à l'utilisateur de cliquer sur la réponse qu'il choisit.

Organisations des données

On travaillera avec des données situées dans deux fichiers `population.txt` et `questions.txt`. Voici un aperçu d'exemple raisonnable pour les questions :

```
Votre personnage fait-il de la politique ?
Votre personnage est-il un homme ?
Connaissez-vous votre personnage personnellement ?
Votre personnage est-il enseignant ?
Votre personnage est-il un acteur ?
Votre personnage est-il blond ?
Votre personnage est-il fictif ?
Votre personnage a-t-il eu une marionnette aux guignols de l'info ?
```

Pour ce fichier de 8 questions (`questions.txt`), on peut imaginer un petit fichier de personnages (`population.txt`) où une ligne donne le nom du personnage puis la ligne suivante donne la liste des réponses attendues aux questions associées.

Un code relativement simple pour les réponses consiste à encoder les réponses avec des entiers :

- 0 : pour la non connaissance de la bonne réponse à la question,
- 1 : pour une réponse positive (sûrement),
- 2 : pour une réponse probablement positive (probablement oui),
- 3 : pour une indécision (ne sais pas),
- 4 : pour une probable infirmation (probablement non),
- 5 : pour une infirmation (sûrement pas),
- -1 : indique la fin du vecteur de réponses associé au personnage.

```
Nicolas Borie
5 1 4 0 5 5 5 4 -1
Stefan Derrick
0 1 5 5 4 4 2 5 -1
Tinky Winky (teletubbies)
0 4 5 0 3 4 1 5 -1
Babar (le roi des elephants)
2 2 0 4 5 0 1 5 -1
Columbo
4 1 0 4 4 0 2 0 -1
Dora l'exploratrice
0 0 5 4 4 5 1 0 -1
Lucky Luke
0 1 4 5 0 5 1 5 -1
Mick Jagger
2 0 5 5 5 0 5 4 -1
Jacques Chirac
5 1 5 0 0 4 5 1 -1
```

Il est clair que plus la base de données est riche, plus le programme est efficace. L'astuce est donc d'améliorer les données au fur et à mesure que le programme est utilisé. Si le programme trouve juste, l'utilisateur a potentiellement donné des réponses non connues du

fichier `population.txt`. Alors on rajoute ces réponses en transformant les `0` par des réponses renseignées `1-5`.

Lorsque le programme se trompe, on propose à l'utilisateur de rajouter une question avec sa réponse correspondante pour son personnage. Une nouvelle question peut alors être rajoutée dans le fichier `question.txt`. Un nouveau personnage peut aussi être rajouté dans le fichier `population.txt`. Lorsqu'une nouvelle question est rajoutée, toutes les réponses correspondantes sont initialisées à zéro sauf pour le nouveau personnage de l'utilisateur.

Stratégie de recherche

Le programme a pour objectif de faire sa meilleure proposition parmi les réponses possibles incluses dans son fichiers `population.txt`. Une stratégie basique pourrait consister en sélectionnant de manière aléatoire les questions que l'on pose à l'utilisateur. Comme l'on peut faire mieux, il faut faire mieux.

Imaginez vous devant un plateau de jeu avec seulement 50 personnages restant qui sont tous les filles (les autres ont été éliminés grâce à des questions précédentes). Parmi ces personnages féminins, 25 sont fictifs et 25 sont réels. Dans ce cas particulier, la question : «Votre personnage est-il une fille ?» n'est d'aucune utilité. Alors que la question «Votre personnage est-il réel ?» devrait vous permettre d'éliminer la moitié des personnages restants. Cette stratégie doit vous rappeler la recherche dichotomique (mis à part qu'ici 5 réponses sont possible au lieu de 2).

Alors comment procéder suite à cette remarque ?

Au début du programme, on initialise une note aux personnages du fichier `population.txt`, une note de 0 est tout à fait raisonnable. À chaque réponse de l'utilisateur, on modifie les notes suivant que les personnages ressemblent aux réponses (en rajoutant des points) ou s'en écartent (en enlevant des points). On établit une barre de sélection en dessous de laquelle les personnages sont considérés comme mauvais (par exemple, tout personnage ayant une note inférieure à moins 10 est considéré hors du jeu (comme si on avait baissé son image associé dans le jeu «Qui est-ce ?»)).

Maintenant, pour déterminer la prochaine question, celle qui permet de raffiner au mieux la sélection, on regarde, pour chaque question encore disponible (c'est à dire pas encore posée), celle qui fait des paquets de tailles les plus équitables. Si 100 personnages sont encore en lice et qu'une question appliquée à ce groupe partitionne 20 personnages pour la réponse 1 (sûrement), 20 personnages pour la réponse 2 (probablement oui), 20 pour la réponse 3, 20 et 20 personnages pour les réponses 4 et 5, alors la question est parfaite et va permettre de se rapprocher grandement du choix de l'utilisateur. C'est la question qu'il faut poser à ce moment précis du jeu.

Une question parfaite n'existe pas forcement, voire, il peut y en avoir plusieurs qui forment des paquets distincts de tailles équitables et alors il faut en tirer une question au hasard parmi les meilleures. Une stratégie intéressante et facile à mettre en place consiste à donner une note à chaque question et a sélectionner la question (ou une des question ayant la meilleure note). Voici une fonction *note* pour noter une question Q de manière raisonnable:

$$note(Q) := \prod_{i=1}^{5} (\#\{p \text{ encore en lice } | Q(p) = i\} + 1)$$

où $\#\{p \text{ encore en lice } | Q(p) = i\}$ désigne le nombre de personnages encore en lice tel que la réponse attendue pour la question Q serait i.

Voici un tableau avec quelques exemples de questions avec leurs notes possibles pour 50 personnages encore en lice :

Question	nb rep. 0	nb rep. 1	nb rep. 2	nb rep. 3	nb rep. 4	nb rep. 5	note
Q_1	22	5	3	0	1	19	960
Q_2	10	19	1	0	1	19	1600
Q_3	11	8	9	7	6	9	50400
Q_4	0	27	23	0	0	0	672

$note(Q_1) = (5+1) * (3+1) * (0+1) * (1+1) * (19+1) = 6 * 4 * 2 * 20 = 960$

$note(Q_2) = (19+1) * (1+1) * (0+1) * (1+1) * (19+1) = 20 * 2 * 2 * 20 = 1600$

$note(Q_3) = (8+1) * (9+1) * (7+1) * (6+1) * (9+1) = 9 * 10 * 8 * 7 * 10 = 50400$

$note(Q_4) = (27+1) * (23+1) * (0+1) * (0+1) * (0+1) = 28 * 24 = 672$

Dans ce contexte où l'on doit choisir parmi 4 questions celle qui départagera au mieux les personnages restants, c'est la question numéro 3 qui obtient la meilleure note. Pour cette question, même s'ils existent 11 personnages encore en lice pour lesquels on ne connaît pas la réponse (les 0), les 39 restants forment des paquets à peut près de même taille. Quelle que soit la réponse donnée par l'utilisateur, des personnages prendront des points et d'autre en perdront. Ainsi, on se rapprochera de la pensée de l'utilisateur.

Passons maintenant à un autre problème classique du développement d'application : l'utilisateur est bête !

Comment toujours deviner les pensées de l'utilisateur alors qu'il peut se tromper de réponse sur quelques questions ?

Il ne s'agit pas ici de tout corriger, l'utilisateur qui ne joue pas le jeu correctement va envoyer le programme dans les choux. Toutefois, en programmant de manière fine les évolutions de notes des personnages, on peut facilement deviner un personnage en 20 questions même si l'utilisateur se trompe sur 2 ou 3 d'entre elles.

L'idée est ici de donner des points positifs ou négatifs aux réponses qui se ressemblent. Les réponses 1 (sûrement) et 2 (probablement oui) se ressemblent un peu, on peut imaginer donner quelques point positifs si l'utilisateur donne une réponse proche de celle attendue. Une réponse 3 (ne sais pas) pour une réponse attendue 5 (sûrement pas) ne peut donner que quelques points négatifs alors qu'une réponse de 1 (sûrement) pour une réponse attendue de 5 (sûrement pas) va faire tomber largement la note du personnage associé.

C'est à vous de choisir des correctifs de note suivant les réponses données et attendues. Pour bien fonctionner, votre programme devra toutefois suivre l'heuristique suivante :

rép. donnée \ rép. attendue	1	2	3	4	5
1	+ + +	+	-	- -	- - -
2	+	+ + +	+	-	- -
3	-	+	+ + +	+	-
4	- -	-	+	+ + +	+
5	- - -	- -	-	+	+ + +

Interface graphique

Ce n'est pas une obligation totale mais c'est tellement mieux. Une bonne interface graphique sera clairement un plus pour l'évaluation de votre projet. Cette dernière devra être développée avec la bibliothèque graphique C de l'université : la libMLV. Une large documentation à propos de cette bibliothèque est accessible à l'URL :
`http://www-igm.univ-mlv.fr/~boussica/mlv/`
Les machines de l'université en sont équipées.

Modularité

Votre programme est une couche algorithmique sur une base de données. Votre programme ne doit pas être dépendant de votre base de données personnelle. Ainsi, vous devez absolument respecter scrupuleusement les formats des fichiers `questions.txt` et `population.txt`. Des scripts analyseront votre programme sur une base de données test (seuls les correcteurs y ont l'accès). Pour vérifier la modularité de votre application, il est très vivement conseillé (pour ne pas dire obligatoire) de tester votre exécutable avec les bases de données d'autres groupes.

Même si une large base de donnée est un plus, sa taille ne sera pas un critère déterminant dans l'évaluation. Il ne s'agit pas de jouer toute la journée pour apprendre un maximum de choses à votre programme. Il faut tout d'abord un programme intelligent qui pose les bonnes questions.

Pour les questions, il faut absolument une question par ligne. Le fichier `population.txt` correspondant doit obligatoirement contenir des vecteurs de réponses de longueur égale aux nombre de questions avant de terminé par la valeur -1.

Pour aller plus loin

Pour les plus récalcitrants d'entre vous, toute amélioration sera considérée comme un plus pour l'évaluation. Voici quelques suggestions possibles de raffinements pour ce projet :

- Rajouter une image pour les personnages de votre base de données (attention à la taille et aux droits des images). Si vous faites de la sorte, utilisez des images de type avatar de taille raisonnable.
- Proposer à l'utilisateur de continuer sur une seconde série de questions lorsque la machine n'a toujours pas trouvé après 20 questions.
- Tentez d'évaluer dynamiquement le rapprochement vers les pensées du l'utilisateur. Le programme peut proposer sa réponse en indiquant à quel point il est sûr ou pas de sa proposition, voire, question après question, le programme peut donner un indice sur sa progression.
- Si le programme est sûr d'avoir trouvé la bonne réponse au bout de 15 questions, il peut utiliser ces 5 dernières questions pour remplacer les 0 de la base de données (et donc enrichir ces connaissances...)
- Proposer une solution semi-humaine pour faire la fusion de deux base de données. Seul un humain peut dire si deux questions sont les mêmes ou pas, après la fusion de réponses renseignées peut être automatisée.

- Garder un historique sur les parties jouées. Le nombre de parties, le nombre de personnage bien devinés, le nombre de personnages rajoutés, le taux de remplissage de réponses correctement renseignées...
- Toute idée qui ne soit pas mauvaise. Attention, le correcteur pourrait avoir un avis différent du votre. Le consulter au préalable est un plus pour ne pas développer inutilement.

Conditions de développement

Le but de ce projet est moins de pondre du code que de développer le plus proprement possible. C'est pourquoi vous développerez ce projet en utilisant un système de gestion de versions Subversion. Le serveur à disposition des étudiants de l'Université est disponible à cette adresse : `https://reposetud.univ-mlv.fr/`. Comme nous attendons de vous que vous le fassiez sérieusement, votre rendu devra contenir un dump du fichier de logs des opérations effectuées sur votre projet, extrait depuis le serveur de gestion de versions ; afin que nous puissions nous assurer que vous avez bien développé par petites touches successives et propres (commits bien commentés), et non pas avec un seul commit du résultat la veille du rendu. L'évaluation tient compte de la capacité du groupe à se diviser équitablement le travail.

Remarques importantes :

- L'intégralité de votre application doit être développée exclusivement en langage C (la documentation technique dynamique fait évidemment exception). Toute utilisation de code dans un autre langage (y compris C++) vaudra ZÉRO pour l'intégralité du projet concerné.
- En dehors des bibliothèques standards du langage C et de la libMLV, il est interdit d'utiliser du code externe : vous devrez tout coder vous-même.
  Toute utilisation de code non développé par vous-même vaudra ZÉRO pour l'intégralité du projet concerné.
- Tout code commun à plusieurs projets vaudra ZÉRO pour l'intégralité des projets concernés.

Conditions de rendu :

Vous travaillerez en binôme et vous lirez avec attention la Charte des Projets. Il faudra rendre au final une archive `tar.gz` de tout votre projet (tout le contenu de votre projet `svn`), les sources de votre application et ses moyens de compilation. Il sera alors crucial de lire des recommandations et conseils d'utilisation de `svn` sur la plate-forme `moodle` (à venir). Vous devrez aussi donner des droits d'accès à votre chargé de TD et de cours à votre projet via l'interface `redmine`.

Un exécutable `mydevin` devra alors être produit . Naturellement, toutes les options que vous proposerez (ne serait-ce que –help) devront être gérées avec `getopt` et getopt_long.

La cible `clean` doit fonctionner correctement. Les sources doivent être propres, dans la langue de votre choix, et commentées. C'est bien de se mettre un peu à l'anglais si possible.

Votre archive devra aussi contenir :

- Un fichier `log_dev` correspondant au dump des logs de votre projet (nom des commits, qui? et quand?), extrait depuis le serveur de gestion de versions que vous aurez utilisé.
- Un fichier `makefile` contenant les règles de compilation pour votre application ainsi que tout autre petit bout de code nécessitant compilation (comme les tests par exemple).
- Un dossier `doc` contenant la documentation technique de votre projet ainsi qu'un fichier `rapport.pdf` contenant votre rapport qui devra décrire votre travail. Si votre projet ne fonctionne pas complètement, vous devrez en décrire les bugs.
- Un dossier `src` contenant les sources de votre application.
- Un dossier `include` contenant tous les headers de vos différents modules.
- Un dossier `data` contenant les deux fichiers `questions.txt` et `population.txt` modélisant la base de données.
- Un dossier `bin` contenant à la fois les fichiers objets générés par la compilation séparée.
- Aucun fichier polluant du type `bla.c~` ou `.%smurf.h%` généré par les éditeurs. Votre dossier doit être propre !
- Si possible, la partie html générée par l'utilitaire `doxygen` à partir de votre application de visualisation. Ceci est optionnel mais tellement plus propre.

Sachant que de nombreux vilains robots vont analyser et corriger votre rendu avant l'oeil humain, le non respect des précédentes règles peuvent rapidement avorter la correction de votre projet. Le respect du format des données est une des choses des plus critiques.