Projet C++ - Akinator

Alexandre de Larrard

12 février 2015

Table des matières

1 Présentation du projet

1.1 Sujet demandé

Implementation d'un algorithme convaincant (et éventuellement de l'interface) de classification pour mimer le jeu en ligne Akinator. Akinator s'autorise à poser 20 questions binaires (oui/non) afin de deviner une personnalité à laquelle le joueur pense. On passera par des arbres de classification pour l'algorithme.

1.2 Corps du programme

Le programme C++ a été développé sous l'environnement QT. Notamment l'interface graphique du programme utilise les objets QT. Le programme est composé de 7 classes dont les fonctionnalités sont les suivantes :

- **Main** : uniquement utilisé pour lancer la classe MainWindows contenant les "Widget" nécessaires pour l'interface graphique ;
- **MainWindows** : classe permettant de déclarer les objets graphiques, de les implémenter et de les lier à leurs slots respectifs (principalement des on click() et get Text Area()). Cette classe instancie également l'arbre de classification et gère les réponses aux questions posées au joueur ;
- **Basededonnees** : classe gérant le fichier d'entrée .txt contenant la base de données de personnages. Cette classe est constituée de plusieurs méthodes traitant la base de donnée (ajout de lignes, chargement de la base de donnnées dans un vecteur de pointeurs, etc.) ;
- **Personnages** : classe donnant la structure des attributs d'un personnages de la base de données (18 au total) ;
- **Noeud** : classe donnant la structure d'un noeud de l'arbre de décision ;
- **ArbreClassification** : classe permettant de construire l'arbre de décision à partir d'une base de données ;
- **Questions** : La classe question permet de poser une question à partir d'une variable et d'une modalité.

Les ressources nécessaires à l'entrée du programme sont :

- La base de données akinatorv2.txt
- Les images en .png nécessaires pour l'interface graphique

Remarque : Attention, le programme se lance une fois que le bouton Jouer est sélectionné. Il prend 1 minute environ à compiler donc il ne faut pas fermer avant d'avoir la fin des retours dans la console.

2 Réalisation de l'algorithme d'arbre de décision

2.1 Fonctionnement de la classe Personnages

La classe personnages permet de construire des objets de type personnages constitués de 18 caractéristiques (18 string). Ainsi chaque personne gérée par l'arbre de décision aura une structure de type personnage constituée des caractéristiques suivantes :

- **ID** : identifiant de la personne dans la base de donnée ;
- **Prenom**, **Nom**, **genre**(M/F) ;
- **Age** : discrétisé en 5 classes pour pouvoir gérer cette variable comme un string. Les 5 modalités sont des tranches d'age tous les 20 ans (ex : 20-40) ;
- **vivant** : 1 = oui, 0 = non (gérés comme un string dans le programme)
- **fiction** : 1=oui, 0=non. Cette variable est toujours à 0 dans la base de données car je n'ai pas encore géré les personnages de fiction (pas de données tangibles dessus dans une base de données).
- **celebre** : 1= oui, 0=non. Tous les personnages sont célèbres dans la base de données créée ;
- **pays origine** : pays de naissance du personnage ;
- **Pays residence** : première lettre du prénom du personnage ;
- **domaine métier** : principalement la Politique, le Cinema, le Sport et la Musique ;
- **metier** : le métier du personnage (ex : acteur, chanteur, president, etc.) ;
- **passe metier** : spécificité du métier du personnage (ex : acteur dans une série, chanteur Rock, etc.)
- **proche** : variables non remplie (l'idée était de gérer les personnes non célèbres et proche du joueur) ;
- **enfants** : décennie de naissance du personnage ;
- **groupe** : continent de naissance du personnage ;
- **ecran** : nom du film ou de la série dans lequel le personnage est apparu ;
- **religion** : variable créée pour y insérer un coefficient d'importance suivant si le personnage demandé à une plus forte probabilité d'être demandé qu'un autre. Pour l'instant tous les coefficients sont à 0.

La classe a également pour but de gérer les setter et les getter des caractéristiques (ci-dessus) des personnages.

2.2 Fonctionnement de la classe basededonnees

La classe base de données est constituée de plusieurs méthodes gérant la base de données dont les principales sont celles qui suivent.

GenererBaseDeDonnees Cette méthode ne prend aucune entrée mais à partir du chemin d'accès au fichier .txt contenant la base de données, elle renvoie un vecteur de pointeur de type personnages. Ce vecteur correspond à la base de donnée.

cash Cette méthode prend en entrer un vecteur de personnages (base de données de personnages) et recopie la base de données dans un autre vecteur de personnages. Cela est utile pour la construction de l'arbre de décision.

SupprimerIndividuOuiBdd Prend en entrée la base de données de personnages (vecteur de pointeurs de personnages), la variable étudiée ainsi que la modalité de la variable étudiée. La méthode supprime alors tous les personnages de la base de données **n'ayant pas** la modalité pour la variable.

SupprimerIndividuNonBdd Prend en entrée la base de données de personnages (vecteur de pointeurs de personnages), la variable étudiée ainsi que la modalité de la variable étudiée. La méthode supprime alors tous les personnages de la base de données **ayant** la modalité pour la variable.

Ajouter reponse La méthode prend un personnages en entrée et l'ajoute à la fin du fichier de sortie bdd akinator.txt, de telle manière à ce que le programme apprenne les réponses non trouvées (car non présente dans la base de données de départ).

2.3 Fonctionnement de la classe Noeud

La classe noeud permet essentiellement de déclarer et construire des objets de type Noeud, nécessaires lors de la construction des feuilles de l'arbre de décision. Les Noeuds de l'arbre sont constitués des caractéristiques suivantes :

- Variable : La modalité qui est utilisée pour construire deux noeuds fils à partir d'un père.
- Modalité : variable string utilisée pour savoir quelle est la modalité qui a été utilisées afin de construire deux noeuds fils à partir d'un noeud père. Ainsi, la variable modalité sera la même pour les deux noeuds fils. Cette caractéristique est nécessaire pour les variables ayant plus de deux modalités (ex : groupe $->$ continent du personnage)
- nombre individu : variable int donnant le nombre d'individu de la base de donnée du fils ayant répondu par un oui à la modalité de sélection du père, ou par un non.
- profondeur : int donnant la profondeur de l'arbre. La racine étant à une profondeur nulle.

– Identifiant noeud : int donnant l'identifiant du noeud fils par rapport à un noeud père. Ainsi, si le père à un identifiant noeud = i, alors le fils répondant par l'affirmative à la variable et la modalité du père aura un $identifiantnoeud = 2 \times i$ et le noeuds fils non aura un $identifiantnoeud = 2 \times i + 1$. Cela correspond à une approche matricielle de la construction de l'arbre de classification.

Une tentative non concluante a été réalisée afin de construire des noeuds ayant en plus en entrée, la base de données de personnes répondant aux contraintes des variables et modalités définies avant dans l'arbre, mais le stockage de bases de données multiples (une dans chaque noeud de l'arbre) pose problème.

2.4 Fonctionnement de la classe questions

La classe questions est essentiellement construire pour la méthode poser Question qui permet de renvoyer un string (output) correspondant à la question à poser au joueur, à partir d'une variable et d'une modalité en entrée. La réponse à la question permet alors de savoir vers quel noeud fils se diriger (à gauche si la réponse du joueur est oui, à droite sinon).

2.5 Fonctionnement de la classe arbre de classification

La classe arbre de classification est la plus complexe car elle crée l'ensemble de l'arbre (vecteur de pointeur de nœuds) à partir de la base de données d'entrée (vecteur de pointeur de personnages).

La construction de l'arbre se fait de la manière suivante :

Initialisation L'arbre est initialisé par un noeud racine constitué de la variable **Genre**. La question est alors : Etes-vous un(e) homme/femme ? La raison d'une telle question vient du fait qu'il y a globalement autant d'homme que de femme dans la base de données de départ, ce qui permet donc bien de séparer au mieux la base de données en une question (cf plus loin sur le critère de sélection de la variable discriminante pour les noeuds fils).

Ainsi à partir du noeud racine, deux noeuds fils sont créés. Le noeud fils de gauche correspondant au Noeud Oui et le Noeud Non à celui de droite. Les deux noeuds sont alors ajoutés à l'arbre pour créer un vecteur de 3 pointeurs de noeuds (1 noeud racine et deux noeuds fils associés).

On a alors l'entier $\underline{tampon} = 2$ car il reste deux noeuds à étudier dans l'arbre. Cette variable permettra de sortir de la boucle while lorsque tous les noeuds de l'arbre seront déclarés comme terminaux, c'est à dire que tous les noeuds de profondeur maximale auront un unique personnage vérifiant ses caractéristiques. Cet unique personnage sera alors la réponse proposée au joueur.

Boucle while Restriction base de donnée :

Dans un premier temps on réinitialise la base de donnée "personnes" via la méthode **cash** de la classe *basededonnnées* afin d'avoir une base de donnée comme neuve (constituée de 650 individus). Cela permet alors de faire appel à la méthode **Restriction bdd** qui prend la base de donnée neuve appelée personnes, et qui la réduit par backward induction. Cela signifie qu'on commence au dernier noeud de l'arbre, et on remonte de père en père. si indice est paire, alors c'est un oui pour la question du père est alors on supprime de la base de données toutes personnes ayant dit non, sinon c'est un non et on supprime tous les oui grâce à la méthode **SupprimerIndividuNonBDD** de la classe *basededonnées*. On se retrouve ainsi avec une base de données constituées des personnages ayant répondu de telle manière à se retrouver dans le noeud étudié.

Il est à noter que la variable indice diffère de la variable identifiant nœud. Cela vient du fait que indice correspond au ième pointeur du vecteur de pointeur de noeud (arbre), alors qu'identifiant noeud est une caractéristique du noeud valant 2∗identifiant noeud du père si c'est un noeud Oui, 2∗identifiant noeud du père +1 si c'est un noeud Non. Le décalage provient du fait que certains noeuds sont terminaux avant d'autres.

Test de terminalité : Si la base de donnée liée au noeud étudié est de taille inférieure ou égale à 1 alors le noeud est terminal, sinon on cherche une variable de sélection.

Sélection de la variable pour les futurs noeuds fils du noeud étudié : Pour chaque variable de la base de données, on cherche la variable (méthode **SelectVariable**) et la modalité (méthode **CreationNoeudsFils**) qui crée deux fils ayant un nombre d'individu aussi proche de 0.5 que possible (méthode **BestSplit**). Comme l'arbre est binaire, cela revient à diviser au mieux la base de données à partir d'un critère simple.

Ajout des nouveaux noeuds fils à l'arbre : Une fois la variable sélectionnée, on crée les deux noeuds fils associés en mettant en entrée la variable sélectionnée ainsi que sa modalité, puis on les ajoute à l'arbre via un push back. Ensuite on incrémente la variable déterminant le noeud étudié (ieme node) de 1 pour passer au noeud suivant de l'arbre dont il faudra étudier les fils optimaux.

On étudie alors le ième node +1 et on recommence le même procédé à partir du while jusqu'à ce que la variable locale tampon vaille 0. Cela signifie que tous les noeuds de l'arbre sont terminaux.

La méthode **Arbre classification** renvoie alors l'arbre construit comme un vecteur de pointeur de noeuds. On trouve dans notre cas, avec une base de données de 635 lignes que l'arbre est composé de 1269 noeuds et a une profondeur maximale de 14. Cela signifie qu'en 14 questions binaires, tous les individus sont identifiables séparément.

```
variable = Genre   nombre individus = 635modalite 1 = M  noeud numero 1  profondeur numero 0
variable = Genre   nombre individus = 253modalite 2 = F  noeud numero 2  profondeur numero 1
variable = Genre   nombre individus = 382modalite 3 = F  noeud numero 3  profondeur numero 1
variable = PaysOrigine   nombre individus = 124modalite 4 = USA  noeud numero 4  profondeur numero 2
variable = PaysOrigine   nombre individus = 129modalite 5 = USA  noeud numero 5  profondeur numero 2
variable = Groupe   nombre individus = 186modalite 6 = Americain du nord  noeud numero 6  profondeur numero 2
variable = Groupe   nombre individus = 196modalite 7 = Americain du nord  noeud numero 7  profondeur numero 2
variable = Metier   nombre individus = 64modalite 8 = Actrice  noeud numero 8  profondeur numero 3
variable = Metier   nombre individus = 60modalite 9 = Actrice  noeud numero 9  profondeur numero 3
variable = Metier   nombre individus = 65modalite 10 = Chanteuse  noeud numero 10  profondeur numero 3
variable = Metier   nombre individus = 64modalite 11 = Chanteuse  noeud numero 11  profondeur numero 3
variable = Metier   nombre individus = 89modalite 12 = Acteur  noeud numero 12  profondeur numero 3
variable = Metier   nombre individus = 97modalite 13 = Acteur  noeud numero 13  profondeur numero 3
variable = DomaineMetier   nombre individus = 80modalite 14 = Cinema  noeud numero 14  profondeur numero 3
variable = DomaineMetier   nombre individus = 116modalite 15 = Cinema  noeud numero 15  profondeur numero 3
variable = Age   nombre individus = 31modalite 16 = 20-40  noeud numero 16  profondeur numero 4
variable = Age   nombre individus = 33modalite 17 = 20-40  noeud numero 17  profondeur numero 4
variable = Age   nombre individus = 34modalite 18 = 20-40  noeud numero 18  profondeur numero 4
variable = Age   nombre individus = 26modalite 19 = 20-40  noeud numero 19  profondeur numero 4
variable = Age   nombre individus = 31modalite 20 = 20-40  noeud numero 20  profondeur numero 4
variable = Age   nombre individus = 34modalite 21 = 20-40  noeud numero 21  profondeur numero 4
variable = Ecran   nombre individus = 31modalite 22 =   noeud numero 22  profondeur numero 4
variable = Ecran   nombre individus = 33modalite 23 =   noeud numero 23  profondeur numero 4
variable = Age   nombre individus = 43modalite 24 = 40-60  noeud numero 24  profondeur numero 4
variable = Age   nombre individus = 46modalite 25 = 40-60  noeud numero 25  profondeur numero 4
variable = Metier   nombre individus = 49modalite 26 = Chanteur  noeud numero 26  profondeur numero 4
variable = Metier   nombre individus = 48modalite 27 = Chanteur  noeud numero 27  profondeur numero 4
```

2.6 Fonctionnement de la classe MainWindow/Main

La classe MainWindow est celle qui permet au joueur de lancer la partie. L'arbre est totalement créé une fois que le bouton Jouer est cliqué. Une fois l'arbre créé, les questions s'affichent sur l'objet QTextEdit à partir des variables et modalités des noeuds de l'arbre, données à la méthode **poserQuestion** de la classe *questions*. C'est à dire qu'au fur et à mesure des questions et des réponses du joueur, on descend dans l'arbre via les noeuds correspondants. A chaque action du bouton Oui ou Non, la base de données personnes est alors réduite jusqu'à n'avoir plus qu'un individu.

Une fois que la base de données n'a plus qu'un individu, le programme pose la question : "Votre personnage est il ...". Si la réponse est bonne la partie peut être relancée par le bouton Jouer. Sinon, le programme demande au joueur d'inscrire dans deux QTextEdit le nom et le prénom du personnage (avec un jeu de setVisible() pour les boutons et objet graphiques). La réponse est alors inscrite dans la base de données bdd akinator.txt via la méthode **AjouterReponse**.

3 Principaux problèmes rencontrés et solutions choisies

3.1 Réalisation de la base de données

Le premier important défi a consisté à trouver une base de données de personnes connues. Malheureusement je n'ai pas pu en trouver une sur internet. J'ai alors réalisé plusieurs programme en python pour extraire les informations de personnalités célèbres du site : http ://www.stars-portraits.com/fr/star/. Les programmes python sont en pièce jointe sous les noms : akinator bdd

suite.py , new akinator bdd suite fin.py. J'ai ainsi pu récupérer les informations sur près de 700 individus avec uniquement 7 variables. J'ai alors dû écrire un second programme python afin d'extraire les informations entre les balises html ¡p¿¡/p¿. Les programmes sont réalisés "à la mano", mais comme ils ne devaient être exécutés qu'une seule fois je ne m'y suis pas longuement attardé.

J'ai également dû faire face à un problème d'encodage que j'ai dû résoudre tant bien que mal en créant des liste avec des caractères spéciaux codés en Utf8...

3.2 Base de données passée en argument du constructeur de Noeud

J'ai essayé une seconde version afin d'optimiser le temps de compilation qui est de l'ordre d'une minute pour la base de données de 700 lignes. Pour cela, j'ai essayé de construire des Noeuds qui avaient un critère en plus, celui de la base de données des individus respectant les spécificités des noeuds antérieurs de l'arbre. Cependant je n'ai pas réussi à débeuger le programme qui ne semblait pas vouloir créer un vecteur de pointeurs de personnages par noeud. Cela provient sans doute de l'absence de smart pointeur qui permettraient d'éviter d'avoir à gérer la destruction d'objet optimisant ainsi la libération de mémoire.