



JADE

Projet Intelligence Artificielle

« Devine à quoi je pense »

Réalisé par Djénéba Djikiné, Alexandre Bernard et Julien Lafont
EPSSI CSII2 - 2011

1. Analyse du besoin

- a. Cahier des charges initial
- b. Fonctionnalités supplémentaires
- c. Sélection des entités

2. Analyse du projet

- a. Méthode de gestion de projet
- b. Planification
- c. Choix des outils de développement

3. Mise en œuvre du projet

- a. Elaboration des arbres de recherche
- b. Choix de la structure de données
- c. Descriptions des principaux algorithmes
 - i. Algorithme d'inférence simple
 - ii. Algorithme de recherche des « bifurcations »
- d. Implémentation de l'interface graphique

4. Démonstration du jeu Jade

- a. Présentation du jeu d'essai
- b. Déroulement de l'algorithme
- c. Capacité d'apprentissage de l'algorithme
- d. Statistiques

5. Conclusions et perspectives

GLOSSAIRE

Entité : Le terme entité désigne le type d'éléments que notre jeu fera deviner. Par exemple Akinator travaille sur deux entités : les personnages et les objets.

Arbre binaire de recherche : Un arbre binaire est un arbre où chaque nœud a au plus deux fils. Dans notre implémentation, les nœuds correspondent aux questions et les feuilles aux entités recherchées. Le chemin pour accéder à une feuille est appelé branche.

Système expert : un système expert est un outil capable de reproduire les mécanismes cognitifs d'un expert, dans un domaine particulier. Il s'agit de l'une des voies tentant d'aboutir à l'intelligence artificielle.

Moteur d'inférence : Un moteur d'inférence est un logiciel correspondant à un algorithme de simulation des raisonnements déductifs. Un moteur d'inférence permet aux systèmes experts de conduire des raisonnements logiques et de dériver des conclusions à partir d'une base de faits et d'une base de connaissances.

1. ANALYSE DU BESOIN

1.1. Cahier des charges initial

L'objectif du projet est de développer **un système expert** capable de deviner des entités en posant à l'utilisateur une série de questions. Ce concept est inspiré de l'application Akinator ou 20Q.

Réalisé sous forme de jeu, le logiciel devra sélectionner les questions pertinentes qui permettront au moteur d'intelligence artificielle de deviner le plus rapidement possible ce à quoi pense l'utilisateur. Pour cela, il s'appuiera sur une base de connaissance composée au démarrage de 100 entités, et d'en moyenne 200 questions, lesquelles seront au fur et à mesure des parties complétées et affinées.

Notre moteur **Jade** (pour **Julien, Alexandre, Djénéba, Epsi**) sera capable de découvrir le sport auquel vous pensez, ou une série télévisée. A chaque étape, des données statistiques seront enregistrées pour améliorer le système, et à la fin de la partie, en cas d'erreur, le logiciel proposera à l'utilisateur de saisir sa réponse et une question discriminative pour ne plus réitérer cette erreur dans les prochaines parties.

1.2. Fonctionnalités supplémentaires

En plus du fonctionnement de base, nous avons décidé de rajouter plusieurs fonctionnalités au cahier des charges initial.

- En plus des boutons « Oui » et « Non » permettant de répondre aux questions, un bouton « Je ne sais pas » sera présent. Son comportement utilisera les statistiques enregistrées tout au long des parties précédents pour choisir la réponse qui a mathématiquement le plus de chance d'être cette correcte.
- En relation, un tableau affichera les statistiques des différentes entités : le nombre de fois où le moteur d'IA a trouvé la bonne réponse, et le nombre de fois où il s'est trompé.
- Une dernière fonctionnalité a été implémenté, celle-ci prend part dans le formulaire d'apprentissage. Lorsque l'utilisateur renseigne l'entité à laquelle il pensait, le moteur va vérifier si elle n'est pas déjà enregistrée dans la base de connaissance. Si elle existe déjà mais que le moteur s'est trompé, il y a 2 explications :

- Soit le moteur a une base de connaissance erronée
- Soit l'utilisateur s'est trompé dans ses réponses

On part du postulat que la base de connaissance est intègre, et donc que l'utilisateur a commis une erreur dans ses réponses.

On va donc rechercher et indiquer à l'utilisateur sur quelle question il s'est trompé.

Pour cela, il est nécessaire de réaliser un double parcours inversé de l'arbre de recherche.

1.3. Sélection des entités

Dans un premier temps, nous avons décidé des critères qui nous permettront de sélectionner nos entités, c'est-à-dire les « choses » que devra deviner le moteur.

Pour être sélectionnée, une entité doit donc offrir au moins 100 déclinaisons différentes (contrainte du cahier des charges), être facilement caractérisable (facilité d'accès aux informations la concernant sur internet), et si possible être originale.

Une contrainte plus difficile à qualifier est la facilité de création de l'arbre. Pour que le moteur d'IA soit efficace, l'arbre de recherche ne doit pas avoir des branches trop longues, et donc avoir de très nombreuses ramifications.

Au final, nous avons sélectionné les deux entités suivantes :

- Les sports en général
- Les séries télévisées françaises et américaines, diffusées de 2008 à nos jours

2. ANALYSE DU PROJET

2.1. Méthode de gestion de projet

Le choix de la méthode de projet et de l'organisation du groupe a été décidé en fonction des contraintes du projet, la principale étant les délais accordés pour réaliser ce projet, relativement courts.

En conséquence, il nous a semblé primordial d'arriver à séparer de manière efficace les tâches afin de pouvoir les paralléliser entre les membres du groupe, permettant ainsi d'accroître la productivité.

De surcroît, la mise en place de **processus agiles** tels que nous les avons étudiées en cours de Génie Logiciel nous a semblé tout indiquée dans ce projet. Cela a permis non seulement de se familiariser avec ces notions théoriques, tout en nous permettant d'être plus efficace dans notre approche de l'étude et du développement.

Le projet et l'équipe étant de taille réduite, nous n'avons pas mis en place une organisation agile complète (telle que décrite par Scrum par exemple), car cela n'aurait pas été efficace. En revanche, nous avons tout de même tenu à mettre en place plusieurs bonnes pratiques tirées de « l'extreme programming » telles que l'intégration continue, l'appropriation collective du code ou encore les tests fonctionnels afin d'assurer une qualité de code optimale.

2.2. Planification

Comme précisé précédemment, nous avons décidé au début de projet de privilégier au maximum les tâches qui peuvent s'effectuer en parallèle. Cependant, il est primordial que certaines décisions soient prises en commun pour assurer la réussite du projet.

Dans cet objectif, nous avons commencé le projet par deux séances de réflexions communes sur le choix des entités, puis sur le fonctionnement général du moteur d'inférence.

Ensuite, nous avons décidé que chaque membre du groupe se spécialiserait dans un des 2 métiers de l'intelligence artificielle : le **développeur**, et l'**expert**.

Le sujet nous amenant à réaliser une recherche sur deux entités indépendantes, 2 étudiants du groupe ont pris le rôle d'expert, ayant pour objectif la conception de l'arbre de recherche.

Le 3^{ème} étudiant a, quant à lui, été chargé d'implémenter l'algorithme et de l'intégrer au sein d'une interface graphique.

Voici un rétro-planning permettant de visualiser les tâches effectuées :

Développeur	Sélection des entités	Réflexion algorithme et arbre de recherche	Test Base donnée embarquée, Jess + Choix solution technique	Définition des normes
Expert Sports			Sélection des éléments Sports	
Expert Séries			Sélection des éléments Séries	

(suite)

Développeur	...	Développement algorithme Interface graphique	Tests d'intégration Rédaction rapport
Expert Sports	...	Elaboration et optimisation de l'arbre de recherche Sports	
Expert Séries	...	Elaboration et optimisation de l'arbre de recherche Séries	

2.3. Choix des outils de développement

Le choix du langage de programmation s'est naturellement orienté sur le langage libre et open-source **Java**.

Dans un premier temps, une étude technique a été réalisée sur le moteur d'inférence Jess (<http://www.jessrules.com/>) afin de déterminer s'il était ou non adapté à notre cahier des charges.

Jess est un moteur de règle maintenu par le laboratoire Sandia (<http://www.sandia.gov>), offrant une implémentation en JAVA du langage Clips. Jess offre un système de programmation par règle permettant de développer des systèmes experts et des moteurs d'inférence. De ce fait, il répond à notre besoin sur la partie technique.

Cependant, plusieurs points nous ont amenés à choisir une solution alternative : **développer notre propre moteur d'inférence**.

Tout d'abord, Jess est un langage propriétaire, dont les sources sont verrouillées et qui n'offre qu'une documentation très partielle. Bien que le système soit lui-même très complet, le développeur ne peut que très difficilement exploiter tout le potentiel de ce moteur. De surcroît, il nous a semblé très intéressant de coder nous-même notre moteur afin d'en comprendre le fonctionnement interne, et de pouvoir l'adapter plus finement à notre besoin.

Concernant la base de connaissance, celle-ci est actuellement stockée dans une Système de gestion de base de données relationnel (SGBRD) dit « embarquée » : **DerbySQL**. Cette caractéristique nous permet de déployer facilement notre application, la base de données est directement incluse et ne demande pas l'installation ou la configuration d'un service extérieur.

De plus, nous avons utilisé la librairie **JDBC** de Java qui permet d'utiliser de manière transparente n'importe quel SGBDR. Il est alors tout à fait possible de transférer la base de connaissance sur une base de données hébergée en ligne, permettant ainsi à plusieurs utilisateurs de se connecter, de partager et d'enrichir une unique base de données.

Concernant le travail des experts, le logiciel libre **FreeMind** a été fort utile. Sa principale fonction est de permettre l'élaboration de cartes heuristiques, mais il est aussi intéressant de l'utiliser pour concevoir un arbre qui, dans la période de réflexion, se veut très évolutif.

Enfin, afin de faciliter les opérations d'intégration continue, nous avons utilisé le logiciel de gestion de version décentralisé **GIT**.

3. MISE EN ŒUVRE DU PROJET

3.1. Elaboration des arbres de recherche

L'idée était de présenter, pour chaque thème choisi, une centaine d'entités environ.

Il a été décidé qu'on fera deux arbres, un pour chaque thème retenu. Cela est pratique dans la mesure où chaque thème est étudié par une personne différente. L'autre raison de ce choix est que les thèmes n'avaient rien en commun.

Le sport comporte un large panel de disciplines. De ce fait, nous avons pris tous les sports officiellement répertoriés, et nous les avons classés par catégorie. Afin de ne pas dépasser le nombre d'entités voulu, nous avons fait un tri, et gardé les sports les plus connus du public.

Le second thème porte sur les séries télévisées françaises et américaines (quelques anglo-saxonnes) qui sont diffusées sur les chaînes de télévision françaises depuis 2008. Un travail de tri a été également effectué sur ce sujet par rapport aux séries les plus connues.

Une fois que nous avons eu les listes, nous avons noté les caractéristiques de chaque sport et série. Cette recherche sert à déterminer les similitudes entre les objets et les points qui distinctifs.

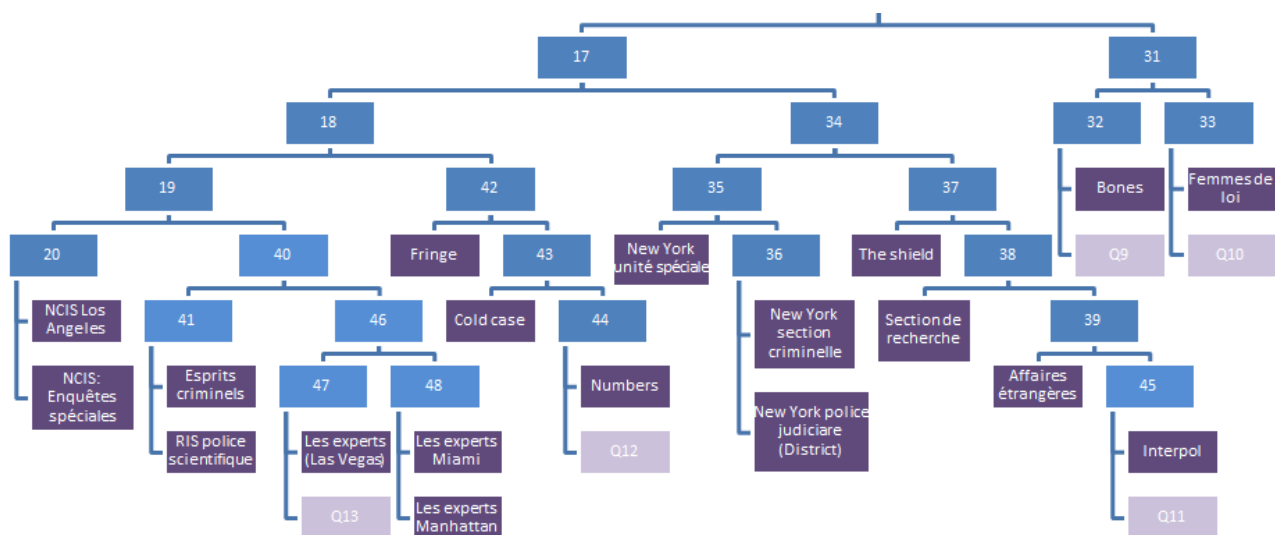
La troisième partie du travail a été la rédaction des questions. Ces dernières découlent directement des spécificités des objets, en s'assurant de ne pas écrire plusieurs fois les mêmes questions. Les questions ainsi obtenues constituent les nœuds de nos arbres.

Après qu'on ait eu nos questions, nous sommes passés à la phase de conception des arbres. Dans un premier temps, nous avons décidé d'utiliser un outil de génération automatique d'arbres de décision comme SIPINA, WEKA, ou encore TANAGRA. Cependant, ces logiciels n'étaient pas adaptés à nos cas. Nous avons finalement jugé mieux de concevoir les arbres nous-même pas à pas.

Nous avons utilisé, pour l'un **Microsoft Word**, pour l'autre **Microsoft Visio** pour élaborer nos arbres.

En effet, il existe des options de représentation d'arbres de toute sorte. L'option permet simplement de voir graphiquement un arbre que nous devons élaborer nous-même.

L'image suivante montre une partie de l'arbre sur les séries télévisées. Les chiffres représentent les questions qui sont numérotées. Les nœuds finaux (sans fils) représentent les séries.



3.2. Choix de la structure de données

La base de connaissance est enregistrée sous la forme d'un arbre binaire.

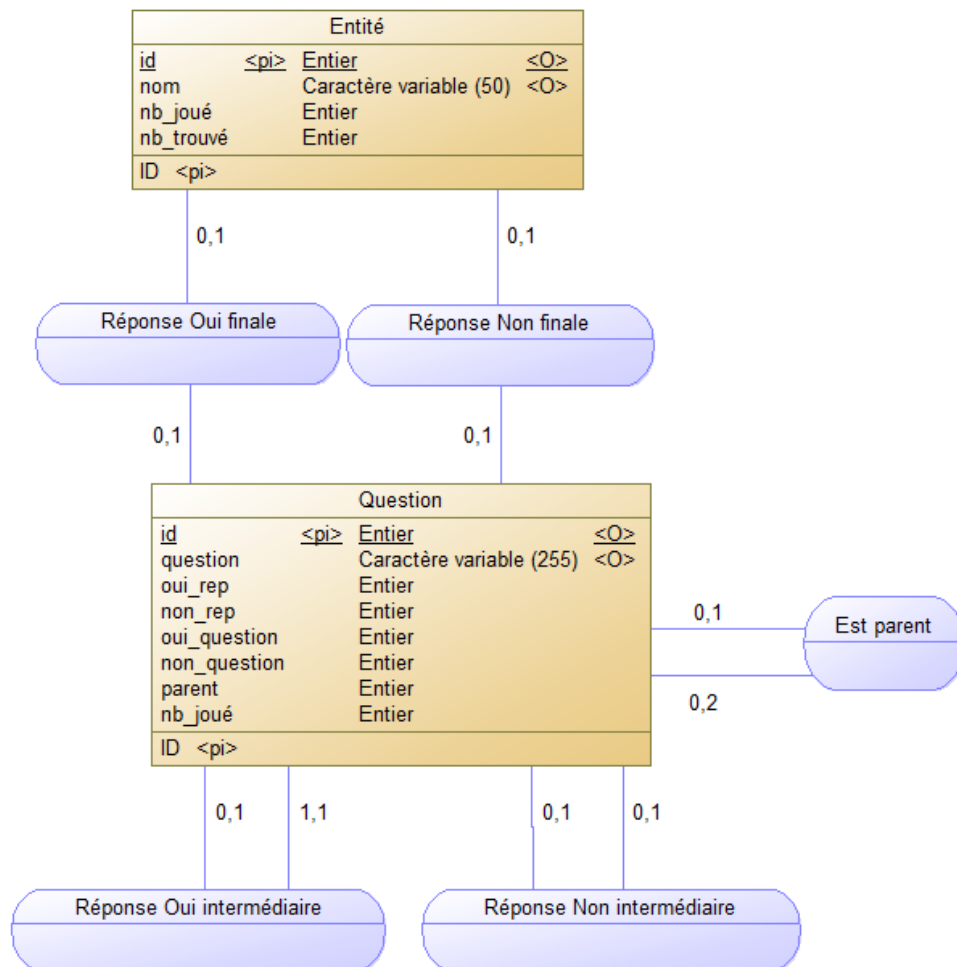
Les données ont été séparés dans 2 tables :

- **Entité** : Les feuilles de l'arbre de recherche
 - Identifiant primaire
 - Nom
 - Nombre de fois où il a été joué
 - Nombre de fois où il a été trouvé

- **Question** : Nœuds de l'arbre de recherche
 - Identifiant primaire
 - Question à poser
 - Réponse OUI
 - ID de la question ou
 - ID de l'entité
 - Réponse NON
 - ID de la question ou
 - ID de l'entité
 - Nombre de fois joué (utilisé pour les statistiques – Réponse « Je ne sais pas »)
 - ID de la question parent : Information dénormalisée pour faciliter la remontée dans l'arbre

Suivant la position de la question dans l'arbre de recherche, pour une réponse donnée (Oui/Non), on définit soit la question suivante, soit l'entité finale.

Voici le MCD correspondant à cette modélisation :



Les entités de types sport et série sont enregistrées dans des tables différentes pour répondre à un besoin d'optimisation et d'évolution. Il sera par exemple possible de rajouter des informations sur la série trouvée (année, producteur, ...) ou sur le sport (record, compétitions, ...) sans rompre le fonctionnement du moteur.

3.3. Descriptions des principaux algorithmes

Dans cette section, nous allons étudier les principaux algorithmes mis en place au sein de l'application.

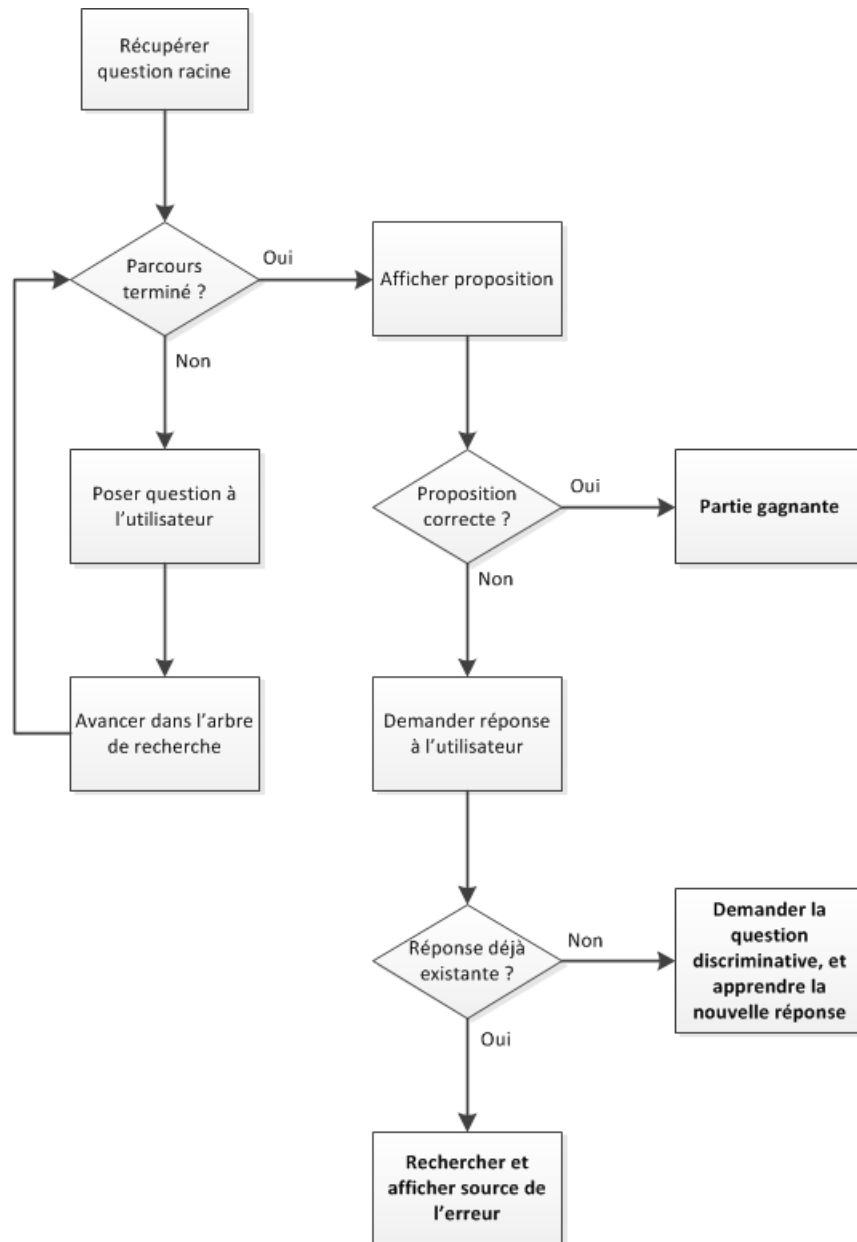
Dans un premier temps, nous analyserons l'algorithme d'inférence principal, c'est-à-dire celui permettant d'avancer dans l'arbre de recherche pour trouver l'entité finale. Dans un second temps, nous verrons l'algorithme permettant de retrouver la question marquant la bifurcation entre deux branches de l'arbre.

ANALYSE DE L'ALGORITHME PRINCIPAL D'INFÉRENCE

Objectif : Trouver l'entité à laquelle pense l'utilisateur en lui posant un minimum de questions fermée.

Stratégie : Parcours avant de l'arbre de recherche préparé par l'expert et amélioré par les précédentes parties du jeu.

Diagramme algorithmique



Programmation algorithmique :

Nœud <- récupérer question racine

Proposition <- Nul

Tant que Proposition est nul **faire**

Poser question à l'utilisateur

Réponse <- Récupérer réponse de l'utilisateur

Nœud <- Avancer dans l'arbre à partir du nœud courant en fonction de la réponse¹

Si nœud est de type question **alors**

question <- Nœud

Sinon

Proposition <- Nœud

Fin si

Fin boucle

Afficher Proposition

Résultat <- Demander à l'utilisateur si la proposition est correcte

Si Résultat est Vrai **alors**

Fin de partie gagnante

Sinon

Réponse <- **Demander réponse à l'utilisateur**

Si réponse existe déjà **alors**

Rechercher source de l'erreur

Afficher à l'utilisateur la question où il s'est trompé

Sinon

Demander à l'utilisateur la question discriminative

Insérer dans la base de connaissance la nouvelle réponse

Fin si

Fin si

1 : Voici une description plus précise de l'action « **Avancer dans l'arbre à partir du nœud courant en fonction de la réponse** ».

Comme nous l'avons vu, pour une question donnée et une réponse donnée (OUI/NON), il existe 2 possibilités :

- Soit la recherche avance vers un nouveau nœud, c'est à dire une nouvelle question
- Soit la recherche avance vers une feuille de l'arbre, c'est à dire une proposition d'entité

De plus, la solution « **Je ne sais pas** » doit être gérée durant cette opération. Dans ce cas, nous récupérons dans un premier temps le nombre de fois où chaque nœud « Oui » et « Non » ont été joués, puis nous retournons le nœud (ou l'entité) qui statistiquement a le plus de chance d'être correcte.

Déclenchement : Cet algorithme est utilisé à la fin d'une inférence infructueuse, lorsque l'utilisateur saisit la réponse à laquelle il pensait et que le moteur détecte que cette réponse est déjà présent dans la base de connaissance. On part du postulat que la base est intègre, on en déduit donc que l'utilisateur s'est trompé à une réponse.

Objectif : Trouver la question à laquelle l'utilisateur a entré la mauvaise réponse.

Stratégie : A partir de deux nœuds de l'arbre (la proposition du moteur et la réponse de l'utilisateur), on remonte récursivement jusqu'à trouver la première question commune.

Optimisation : On parcourt en parallèle les 2 branches de l'arbre et on vérifie à chaque étape si la question parcourue est présente parmi les ancêtres déjà connues de l'autre question de départ.

Programmation algorithmique :

ListeAncetresBrancheA <- { NoeudA }

ListeAncetresBrancheB <- { NoeudB }

Si NoeudA = NoeudB alors

 => Retourner Question du NoeudA

Fin si

Boucler

 Si NoeudA différent de NoeudRacine alors

 NoeudA <- Parent direct de la NoeudA

 ListeAncetresBrancheA <- Ajouter NoeudA

 Si ListeAncetresBrancheB contient NoeudA alors

 => Retourner NoeudA

 Fin si

 Fin si

 Si NoeudB différent de NoeudRacine alors

 NoeudB <- Parent direct de NoeudB

 ListeAncetresBrancheB <- Ajouter NoeudB

 Si ListeAncetresBrancheA contient NoeudB alors

 => Retourner NoeudB

 Fin si

 Fin si

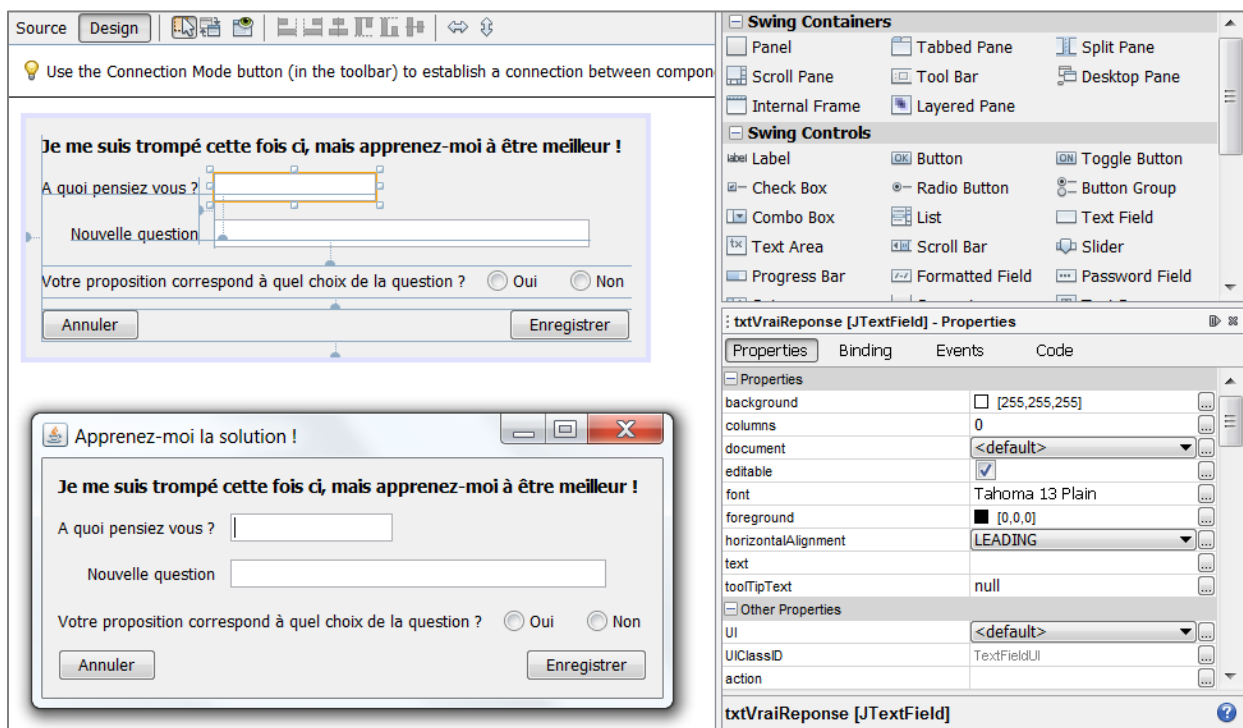
Fin Boucle

3.4. Implémentation de l'interface graphique

Notre choix a été de réaliser une application dite « **client lourd** » pour offrir un fonctionnement proche des jeux commerciaux tels qu'on les connaît habituellement.

La technologie JAVA ayant la force d'être multiplateforme, nous avons accès notre choix sur le framework d'interface graphique Swing qui permet de garder cet avantage.

De surcroît, cette technologie est parfaitement intégrée au sein de l'environnement de développement *Netbeans*, permettant de réaliser facilement les interfaces de notre application.



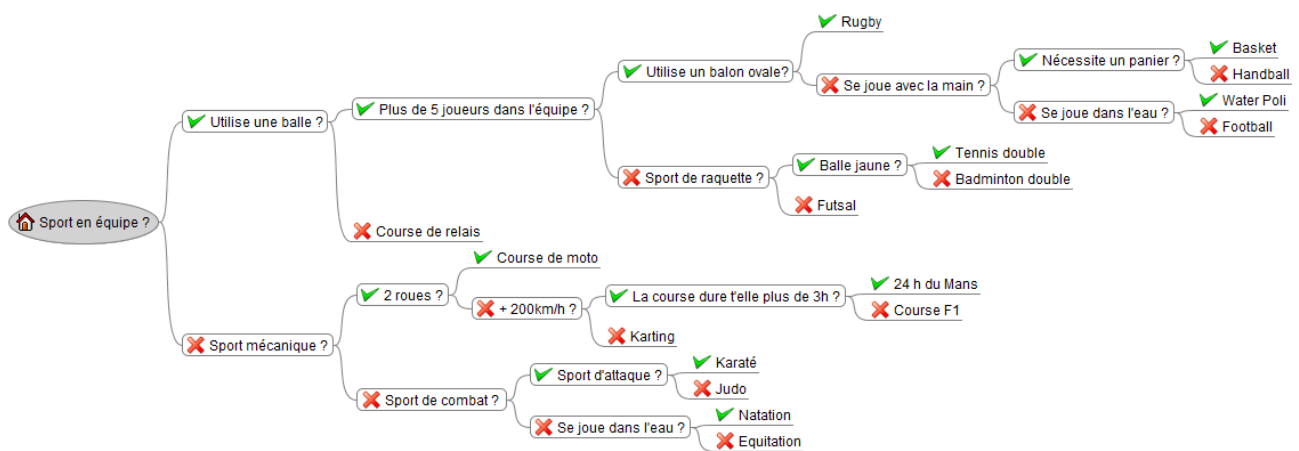
(Développement d'interfaces Swing sous Netbeans)

4. DÉMONSTRATION DU JEU JADE

4.1. Présentation du jeu d'essai

Voici le jeu d'essai que nous allons utiliser pour effectuer cette démonstration de notre jeu.

Il s'agit d'une version très simplifiée de l'arbre de recherche spécialisé dans la recherche des sports.



Le sigle **V** correspond à la réponse Oui
Le sigle **X** correspond à la réponse Non

4.2. Déroulement du jeu

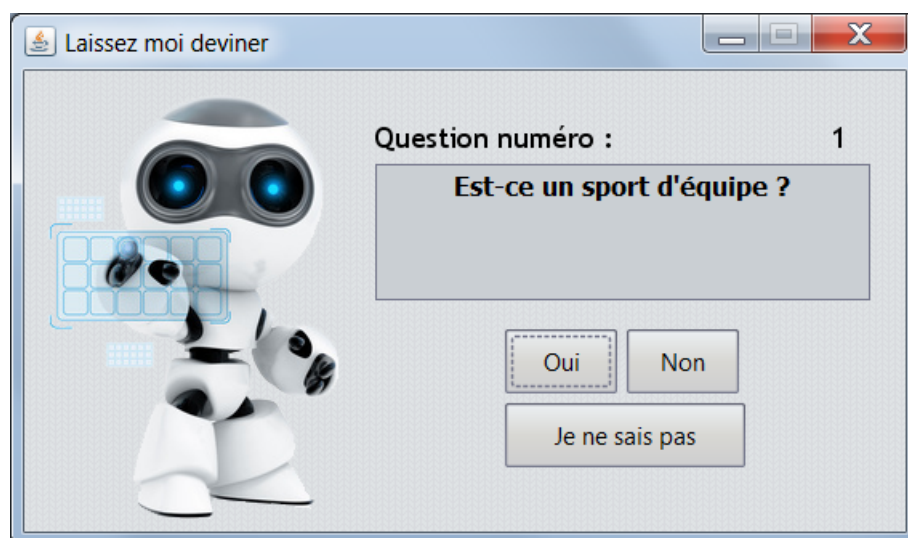
Ecran d'accueil : l'utilisateur est invité à choisir entre les 2 options de jeu :

- Recherche de sports
- Recherche de séries



Inférence en cours : le moteur pose une question à l'utilisateur, à laquelle il peut répondre par « Oui », « Non », ou « Je ne sais pas ».

Cet écran tourne tant qu'aucune proposition n'est retournée par le moteur.



Voici un résumé du déroulement de l'algorithme :

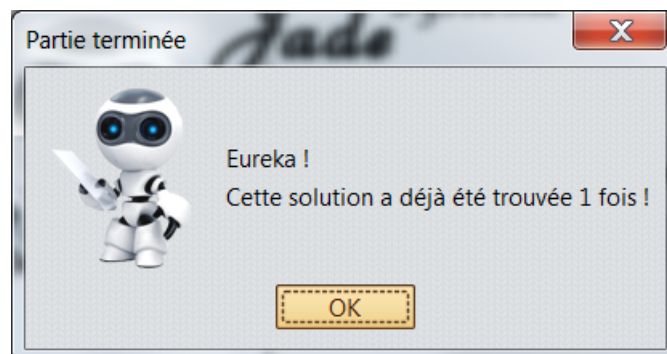
- Est-ce un sport d'équipe ? **Non**
- Est-ce un sport mécanique ? **Oui**
- Les véhicules sont-ils des 2 roues ? **Non**
- Les véhicules peuvent-ils rouler à plus de 200km/h ? **Oui**
- La course dure-t-elle plus de 3h ? **Non**

Proposition du moteur : **Formule 1**

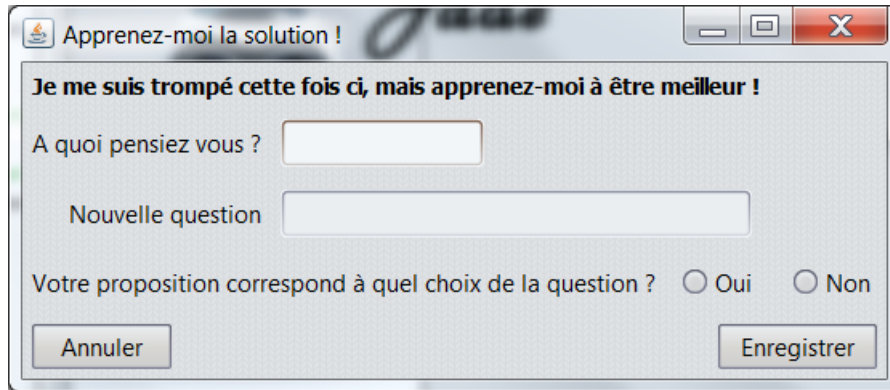


On demande alors à l'utilisateur si la proposition est correcte.

Si le joueur répond **oui**, on affiche un dernier écran qui le félicite et indique le nombre de fois où cette réponse a déjà été trouvée :

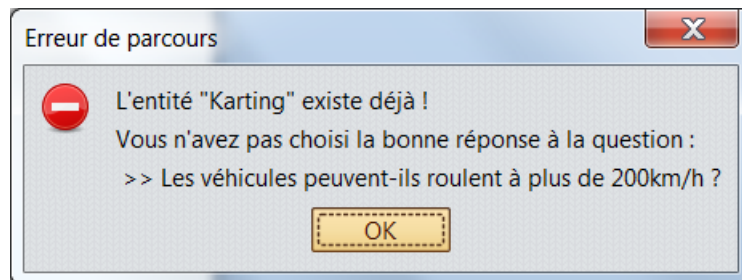


S'il répond non, le moteur va demander à l'utilisateur des informations pour ne plus reproduire cette erreur :



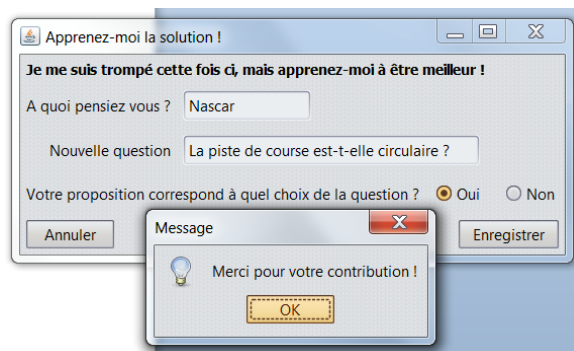
Sur ce dernier écran, si l'utilisateur saisit **une entité qui existe déjà** dans la base de connaissance, le moteur va indiquer sur quelle question il s'est trompé

Voici la fenêtre que l'on obtient si l'utilisateur pensait au « Karting »



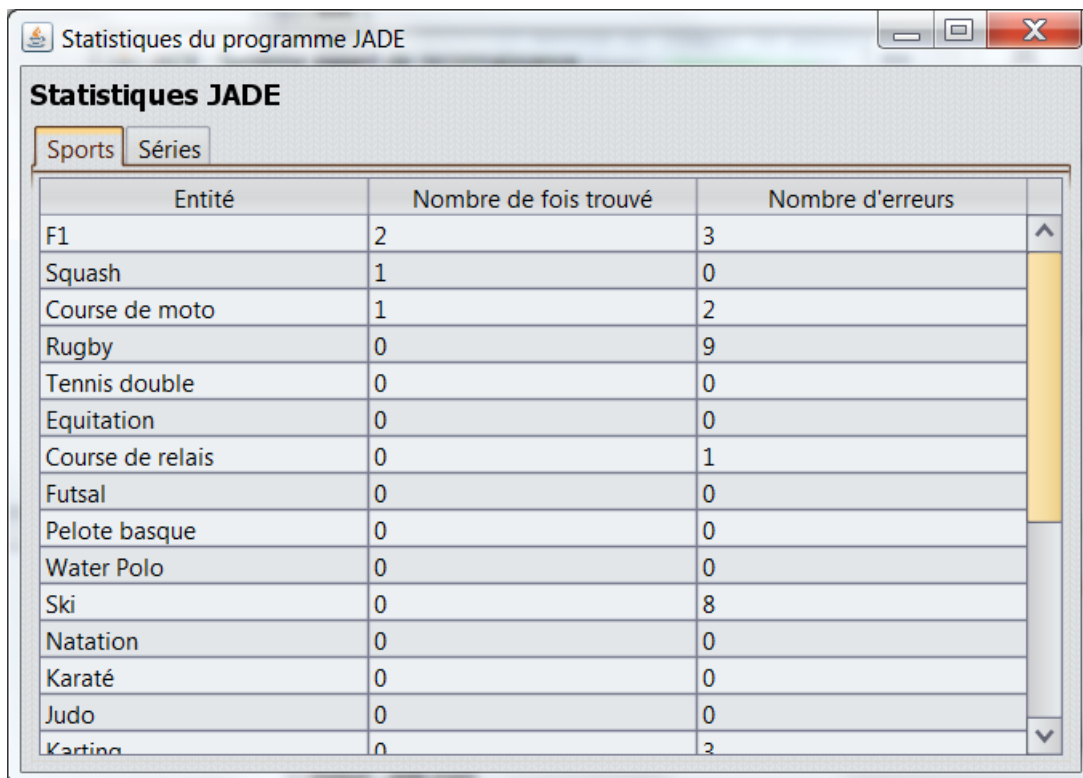
Nous avons répondu Oui, il fallait répondre Non.

Enfin, si l'entité est correcte, le moteur va l'enregistrer dans sa base de connaissance, et elle sera disponible dès la prochaine partie.



STATISTIQUES

L'application fournit un écran récapitulatif permettant d'afficher pour chaque entité enregistrée le nombre de fois où elle a été devinée, et le nombre de fois où le moteur s'est trompé.



Entité	Nombre de fois trouvé	Nombre d'erreurs
F1	2	3
Squash	1	0
Course de moto	1	2
Rugby	0	9
Tennis double	0	0
Equitation	0	0
Course de relais	0	1
Futsal	0	0
Pelote basque	0	0
Water Polo	0	0
Ski	0	8
Natation	0	0
Karaté	0	0
Judo	0	0
Karting	0	3

5. CONCLUSION ET PERSPECTIVES

Au final, l'application Jade fournit un système de recherche parfaitement fonctionnel. La base de connaissance initiale comporte environ 100 sports et séries, avec en moyenne 500 questions par arbre de recherche, et une longueur moyenne des branches de 10-15 questions.

Plusieurs améliorations peuvent être apportées au système pour le rendre plus performant, et le code du moteur et des contrôleurs ont été pensés pour qu'elles soient possibles et facilement intégrables.

La première amélioration serait de transformer l'arbre de recherche en un graphe, où plusieurs branches peuvent se rejoindre dans des cas où la réponse à une question n'est pas formelle. Cela permettrait ainsi de rattraper des erreurs récurrentes des utilisateurs.

Une autre idée est de stocker les questions auxquelles le joueur a répondu « Je ne sais pas », et en cas d'erreur à la fin de la recherche, le moteur pourrait repartir sur ces questions en prenant l'autre réponse. Cela permettrait d'avoir un comportement proche de celui d'Akinator où le moteur a plusieurs chances de trouver la solution.