



BALAX Julien

Licence Informatique 3ème année

Année universitaire: 2011-2012

DebateWEL: Un jeu de dialogue de persuasion utilisant des enthymèmes

Tuteur universitaire: Dupin de Saint-Cyr Bannay Florence

Dates: TER du 09/04/2012 au 09/06/2012

Sommaire

I- Contexte du TER	3
1. Déroulement du TER	3
2. Présentation du cadre	4
A) L'IRIT	4
B) L'équipe ADRIA	5
3. Fonctionnement des réunions de travail	6
II- Sujet	7
1. Résumé/Introduction	7
2. Détail des besoins utilisateurs	8
A) Interface	8
B) Éventail des coups	9
C) Fonctionnalités	10
III- Résolution et approche du problème / Développement	11
1. Spécifications	11
A) Schéma de l'architecture	11
B) Detail des classes	12
2. SAToulouse	15
A) Présentation	15
B) Intérêts pour notre projet	16
C) Intégration/adaptation	17
3. Justifications techniques	20
A) Sauvegarde/Restauration	20
B) Affichage des informations	22
C) Option pour lancement rapide d'une partie	25
D) Difficultés rencontrées	26
IV- Bilan	28
1. Satisfactions	28
2. Regrets	29
A) L'intelligence artificielle	29
B) Le compte-rendu	29
C) Application des méthodes de développement	30
D) Bonnes Conventions de codage	30
V- Conclusion.....	31
VI- Glossaire	32
VII- Annexe	33

1. Déroulement du TER

Nous avons dans un premier temps «dépoussiérer» le sujet en imaginant tous les besoins que le logiciel devait prendre en compte.

Au début du projet, nous en avons trouvé un maximum même si nous savions déjà que certaines des fonctionnalités ne pourraient être explorées.

Mais comme les délais n'étaient pas fixés, il valait mieux en mettre trop que pas assez. Pour ne pas se retrouver avec un logiciel terminé à un mois de la soutenance mais non modulable, non adaptable et avec des fonctionnalités inventées sur le tas.

Après ces deux jours de réflexion, on a passé une semaine de spécification à tenter de trouver l'architecture du projet la plus adéquate pour répondre aux problèmes en utilisant des graphes UML.

Nous nous sommes réunis du lundi au vendredi pendant une heure avec notre référent et chaque lendemain on arrivait avec une évolution de ce qu'on avait proposé la veille mais qu'on avait convenu ensemble de modifier de telle ou telle manière.

Par la suite, la fréquence des réunions s'est réduite ; nous nous rencontrions hebdomadairement.

Nous nous sommes pas réparti le travail de manière très précise.

Lorsqu'un de nous avait terminé sa séance de travail et qu'il jugeait avoir fait avancer le projet, il soumettait une archive du dossier src/ du projet à son camarade par mail (aussitôt suivi d'un sms) en lui faisant un bref rapport de ce qui avait été développé depuis la dernière archive reçu ou envoyé, des difficultés qu'il a rencontré et dont il n'a pas encore trouvé de solution ainsi que ce qu'il aimerait dont on s'occupe dans les jours suivants.

La contrainte principale étant de veiller à ce que l'autre ne code pas en même temps sur les mêmes parties, il était alors indispensable de prévenir son collègue à chaque fois que l'on allait programmer en lui précisant sur quoi on allait travailler et la durée estimée du travail qu'on s'apprêtait à faire.

Il est vrai et je le reconnais qu'au début, quand on n'avait pas encore le squelette du jeu, qu'on était amené à modifier de manière fine tant de classes et tant de fonctions, qu'il s'agissait plus d'un travail en alterné que d'un travail en parallèle.

Mais une fois que l'on a obtenu une version gérant tous les besoins primaires (sans système de restauration, sans gestion du score, deux sortes de coups), on s'est alors réparti le travail de manière plus intelligente (pendant qu'un gèrait l'affichage, l'autre codait un nouveau coup etc.)

2. Présentation du cadre

A) L'IRIT

L'Institut de Recherche en Informatique de Toulouse représente un des plus forts potentiels de la recherche en informatique en France avec un effectif global de 600 personnes dont 250 chercheurs et enseignants-chercheurs, 244 doctorants, 14 Post-Doctorants et chercheurs contractuels ainsi que 43 ingénieurs et administratifs.

Les 19 équipes de recherche du laboratoire sont structurées en sept thèmes scientifiques qui couvrent l'ensemble des domaines de l'informatique actuelle :

- ✓ Analyse et synthèse de l'information
- ✓ Indexation et recherche d'informations

- ✓ Interaction, autonomie, dialogue et coopération
- ✓ Raisonnement et décision
- ✓ Modélisation, algorithmes et calcul haute performance
- ✓ Architecture, systèmes et réseaux
- ✓ Sécurité de développement du logiciel

B) L'équipe ADRIA

L'équipe ADRIA dans laquelle évolue notre tuteur couvre le quatrième thème «Raisonnement et décision».

L'équipe se consacre à différents types de raisonnement :

- ✓ la révision d'informations tenues (plus ou moins certainement) pour vraies à l'arrivée de nouvelles informations
- ✓ la fusion d'informations partiellement contradictoires ou incertaines provenant de sources multiples
- ✓ le raisonnement abductif, qui permet de remonter aux causes plausibles d'une situation observée en diagnostic
- ✓ le raisonnement d'interpolation en logique floue
- ✓ le raisonnement en présence d'exceptions.

Ses travaux portent aussi sur les problématiques de décision, pour lesquelles l'IA développe, avec un souci de calculabilité, des modèles originaux de représentation des préférences, ainsi que de nouveaux critères de décision qualitatifs de façon à être plus proches des manières dont l'humain peut appréhender ses choix.

Une part importante des recherches actuelles de l'équipe concerne la formalisation de l'argumentation. Cette forme de raisonnement permet d'expliquer des conclusions et de gérer des contradictions. Elle joue aussi un rôle crucial dans les dialogues de négociation notamment. Plus récemment, nous avons abordé aussi des questions d'apprentissage, en particulier à partir de données incertaines.

3. Fonctionnement des réunions de travail

Pendant les réunions de travail, nous nous installions tous les trois devant un ordinateur portable équipé de Netbeans.

Nous montrions à l'enseignante les points améliorés depuis la dernière rencontre.

Nous débattions de ce qu'il serait intéressant d'ajouter, de modifier et nous lui présentions ce qu'on pensait concevoir durant les jours à suivre.

Celle-ci nous fixait les priorités mais nous laissait une liberté conséquente.

Nous repartions de chaque réunion avec un compte-rendu faisant office de feuille de bord pour la semaine suivante.

Cette liste n'était pas trop exhaustive, alors une fois que ces points furent gérés on se fixait par e-mail la suite des événements.

1. Résumé/Introduction

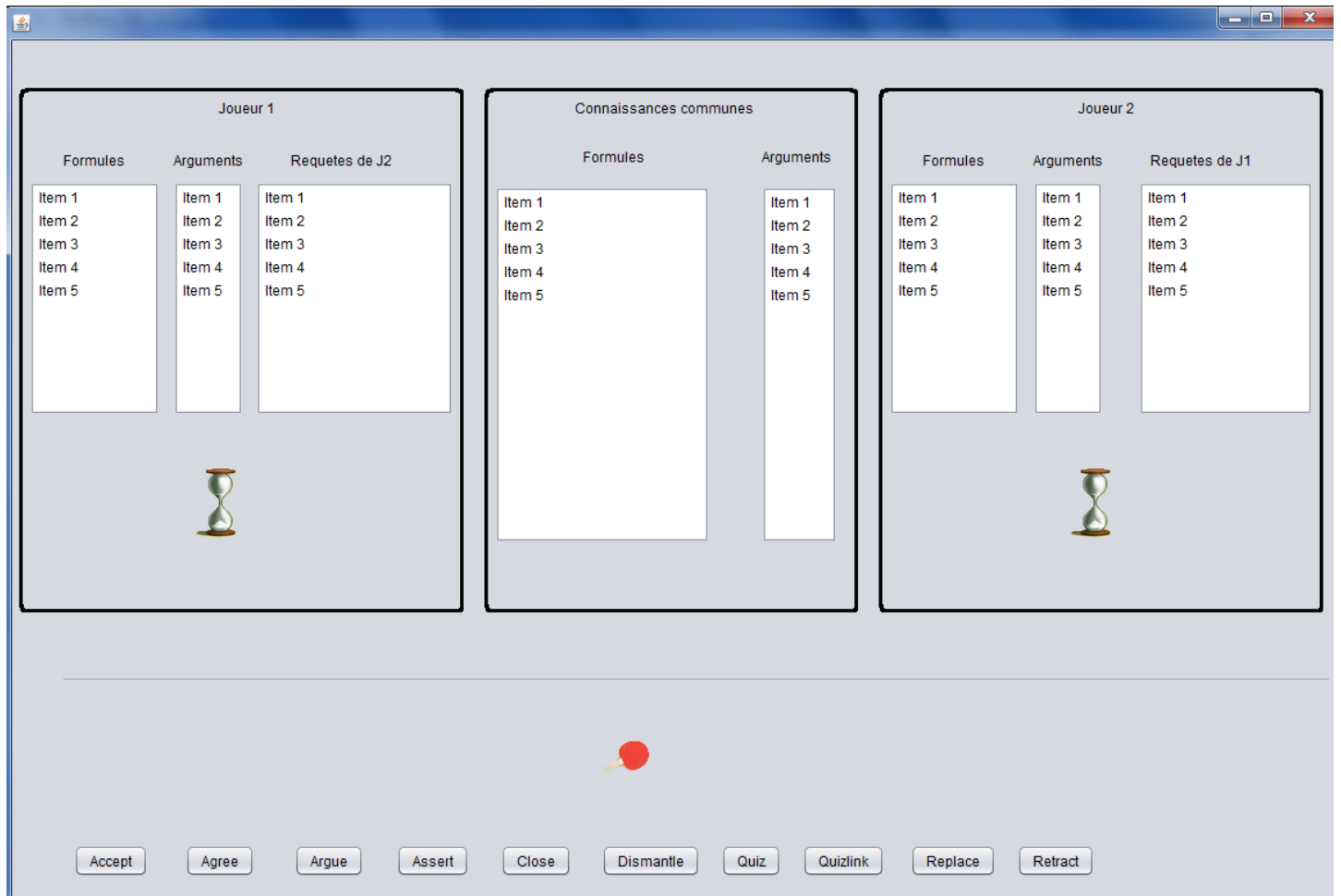
Plusieurs spécificités sont codées dans DebateWel .

- ✓ deux agents sont considérés (par soucis de simplicité)
- ✓ au commencement du dialogue, les agents se mettent d'accord sur un socle de connaissances communes sur lequel ils vont baser leurs prochaines déclarations.
- ✓ pas d'information concernant l'état d'esprit des agents, seulement ce qui sera prononcé publiquement sera pris en compte
- ✓ le contenu des coups est basé sur la logique propositionnelle classique
- ✓ les arguments échangés ne sont pas des entités abstraites mais des paires (S, c) où S est un ensemble de formules (appelée support) et c une formule constituant la conclusion.
- ✓ Par ailleurs, l'idée est d'autoriser la saisie de couples dans lesquels S n'est pas une preuve parfaite minimale de c.
 - ✗ Quand quelque chose manque pour prouver c depuis S, cette paire est appelé un enthymème.
 - ✗ L'utilisation d'enthymèmes est très commune dans les discussions entre individus car la preuve est souvent trop longue à énoncer mais évidemment admise.
- ✓ La connaissance commune pour les deux agents est composée de formules et enthymèmes (car ils peuvent convenir qu'une paire est une preuve imparfaite de sa conclusion bien que suffisamment convaincante).
- ✓ Cette connaissance commune peut seulement s'agrandir au cours du dialogue lorsque les deux agents sont d'accord avec l'adversaire sur une proposition ou un enthymème.
- ✓ Le protocole est totalement flexible, à savoir qu'un agent peut effectuer un coup quand il le souhaite, peut dire ce qu'il veut (à condition qu'il ait la main) tant qu'il n'enfreint pas les trois règles données ci-dessous.
 - ✗ l'auto-contradiction
 - ✗ la répétition
 - ✗ pour fermer le dialogue, l'agent doit remplir tous les engagements engendrés par les mouvements de l'autre agent.
- ✓ Le débat s'achève soit par la victoire d'un agent si l'autre agent est d'accord avec lui ou avec un échec si les agents n'ont pas réussi à remplir leurs engagements dans le temps imparti (soit exprimé en secondes ou en termes de nombre de symboles utilisés).
- ✓ L'idée du debateWEL est d'encourager les agents à se retrouver sur un terrain d'entente plutôt que de parvenir à un échec du débat, d'où un score
 - ✗ élevé pour un agent victorieux
 - ✗ médian pour un agent défait
 - ✗ faible pour un dialogue échoué.

2. *Détail des besoins utilisateurs*

A) Interface

Nous avons dès le départ conçu une interface sans handler pour gérer les événements. Ceci dans le but de valider auprès de notre tuteur les éléments de base que notre jeu devait contenir.



B) Éventail des coups

- X Accept(ϕ): acceptation de la formule ϕ
- X Agree(arg, ϕ): acceptation de l'argument arg éventuellement incomplet qui justifie ϕ
- X Argue(arg, ϕ): annonce d'un argument éventuellement incomplet qui justifie ϕ
- X Assert(ϕ): affirmation de la formule ϕ
- X Challenge(ϕ): demande d'un argument justifiant ϕ
- X Close fermeture du dialogue
- X Dismantle<S, ϕ > : retrait de l'argument <S, ϕ >
- X Quiz<S, ϕ >: demande de complétion de l'argument <S, ϕ >
- X Quizlink<S, ϕ >; 'i demande d'une complétion dont la conclusion

a un lien avec le dialogue

✗ Replace (a; b) annonce d'un argument b qui complète a

✗ Retract(phi): retrait de l'affirmation phi

C) Fonctionnalités

✓ Base d'exemples

✗ On doit avoir la possibilité de charger des fichiers dit thèmes afin d'alimenter les fenêtres de saisie de formules et d'arguments, l'idée étant d'inciter les joueurs à ne pas proposer d'atomes ne figurant pas dans l'ensemble de cette base.

✗ Ceci permettrait de ne pas avoir un débat trop ouvert mais bien un débat qui traite un sujet en profondeur.

✓ Compte rendu

✗ Lorsque la partie s'achève, le logiciel calcule un compte rendu en remplaçant chaque variable propositionnelle par des affirmations tirés d'une base de donnée existante ou créée.

✗ Mieux il pourra essayer de remplacer les inclusions par des mots conjonctions ou des adverbes (ainsi, donc), les et par des « ainsi que » les ou par des « mais aussi » etc.

✓ Mode

✗ Multijoueur Player 1 vs Player 2

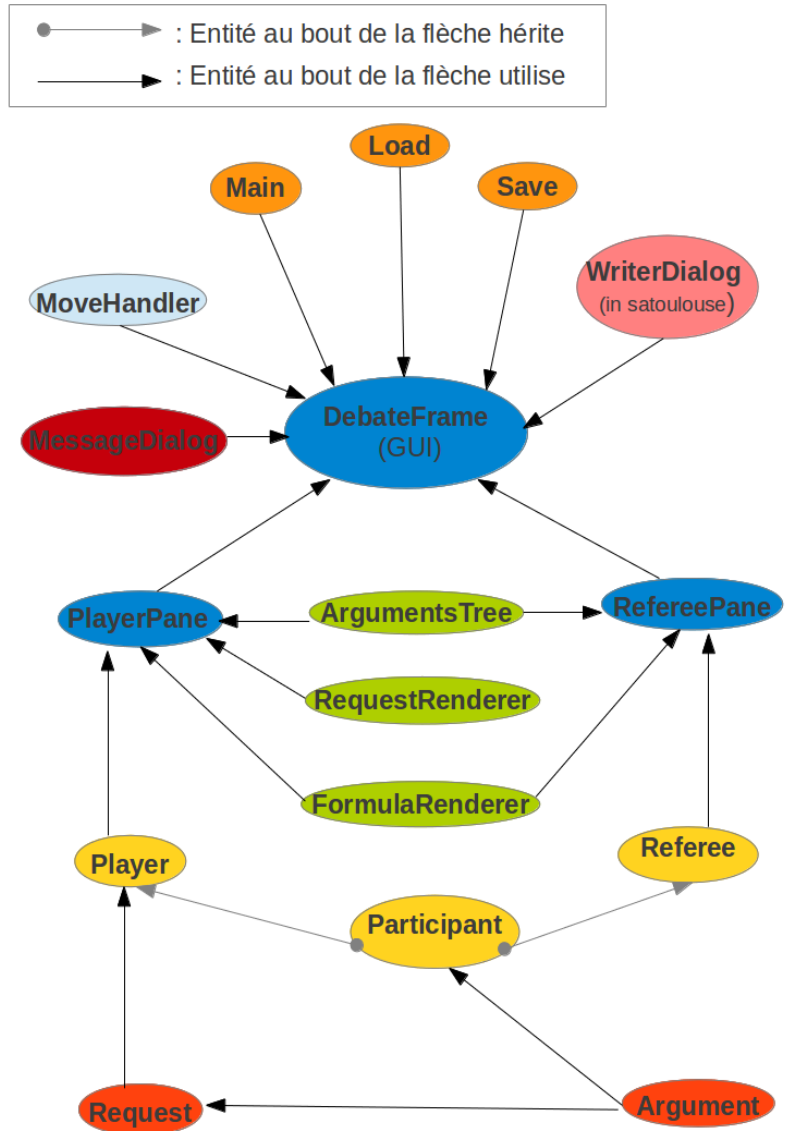
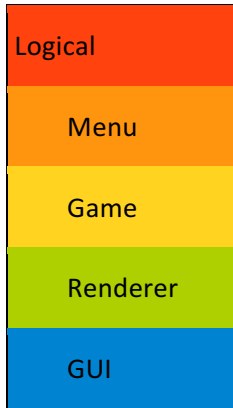
✗ Player vs IA

✓ Système de restauration/sauvegarde

✗ Enregistrement du répertoire de sauvegarde pour l'ouvrir aussitôt comme répertoire par défaut lors d'une restauration.

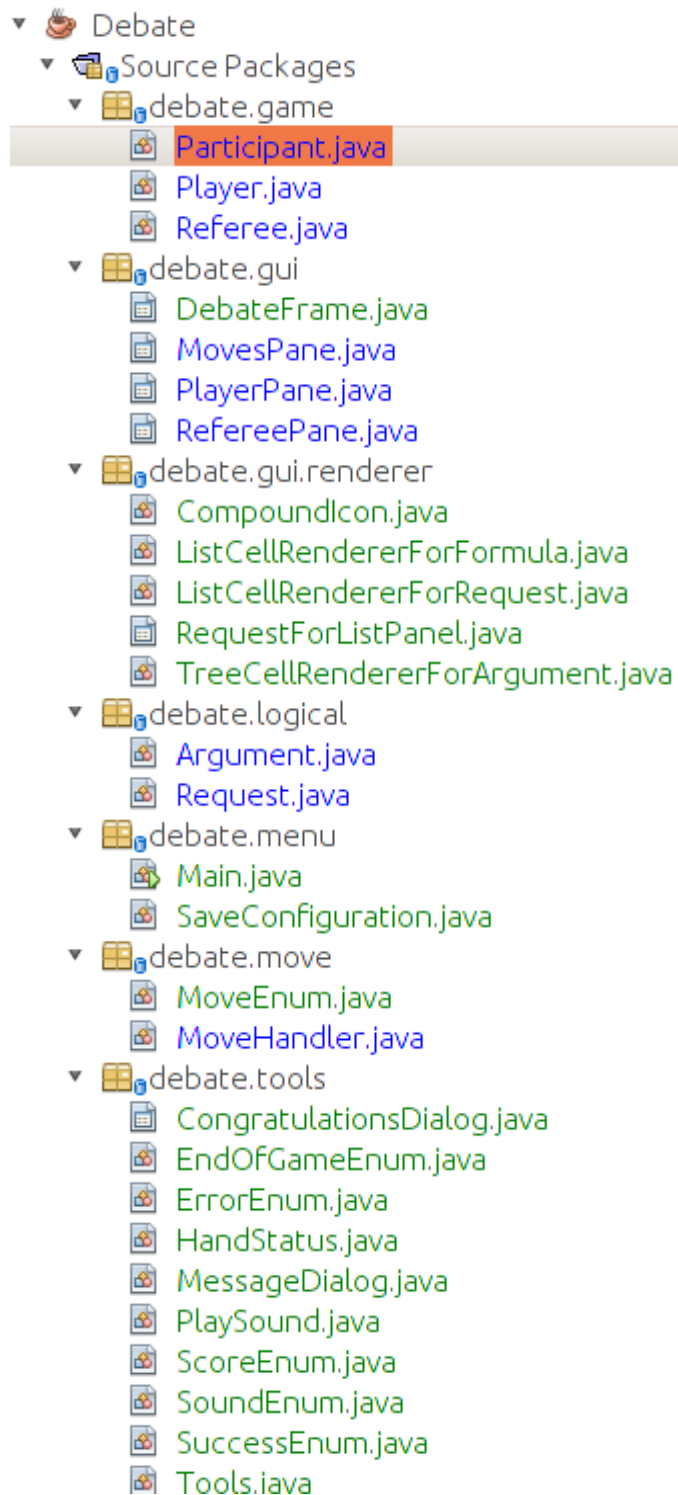
1. Spécifications

A) Schéma de l'architecture



B) Detail des

classes



Participant

Liste des formules: List

Liste des argument: List

Solver : SATSolverSAT4J

Player

(extends Participant)

Liste des requêtes: List

Compteur de formules acceptés: int

	Pseudonyme: string Temps de parole: int
Referee (extends Participant)	Activation du son: boolean. Situation (qui a la main): HandStatus Liste des thèmes chargés:List Compteur des coups acceptés: int.
CompoundIcon	
DebateFrame	Interface principale, GUI. Traite les évènements générés par l'utilisateur.
MovesPane	Panneau comportant les 9 coups réalisables.
PlayerPane	Panneau propre aux informations du joueur. Passerelle entre la GUI et le joueur qu'elle contient.
RefereePane	Panneau propre aux information de l'arbitre. Passerelle entre la GUI et l'arbitre qu'elle contient.
ListCellRendererForFormula	Rendu pour afficher une formule en Latex (explication plus bas)
ListCellRenderForRequest	Rendu pour afficher une requête en Latex avec puce coloré selon le type de la requête
TreeCellRendererForArgument	Rendu d'un argument
Argument	support (liste de formule) conclusion (formule)
Request	<ul style="list-style-type: none"> – formule – argument – type de coup
Main	Lance la GUI DebateFrame en installant le LookAndFeel au préalable
SaveConfiguration	Enregistre les infos dans un fichier texte (voir plus bas la manière utilisée)
MoveEnum	Type de coup (Assert, Accept, Agree etc)
MoveHandler	Vérifie redondance d'une formule ou d'un argument par rapport à la liste du joueur (self redundancy) ou de l'arbitre (global

	redundancy).
CongratulationsDialog	Fenêtre de félicitations
EndOfGameEnum	Type de déclenchement de la fin de partie
ErrorEnum	Type d'erreur à envoyer à MatDialog
HandStatus	Type de situation: personne n'a la main , joueur 1 ou joueur 2
MessageDialog	Message de succes ou d'erreur
PlaySound	Lance le son selon le SoundEnum lui étant passé en paramètre.
ScoreEnum	Type de comptage du score
SoundEnum	Type du son à jouer
SuccessEnum	Type de succes à envoyer à MatDialog

2. SAToulouse

A) Présentation

Ce logiciel permet de s'initier à la programmation logique à l'aide de la logique propositionnelle.

L'utilisateur programme la résolution d'un problème en le décrivant à l'aide de la logique propositionnelle. Contrairement à la programmation impérative (Java, C, etc.), où on donne les instructions qui devront être exécutées pour résoudre un problème, la programmation logique est descriptive.

Programmer revient à décrire les règles du problème que l'on souhaite résoudre.

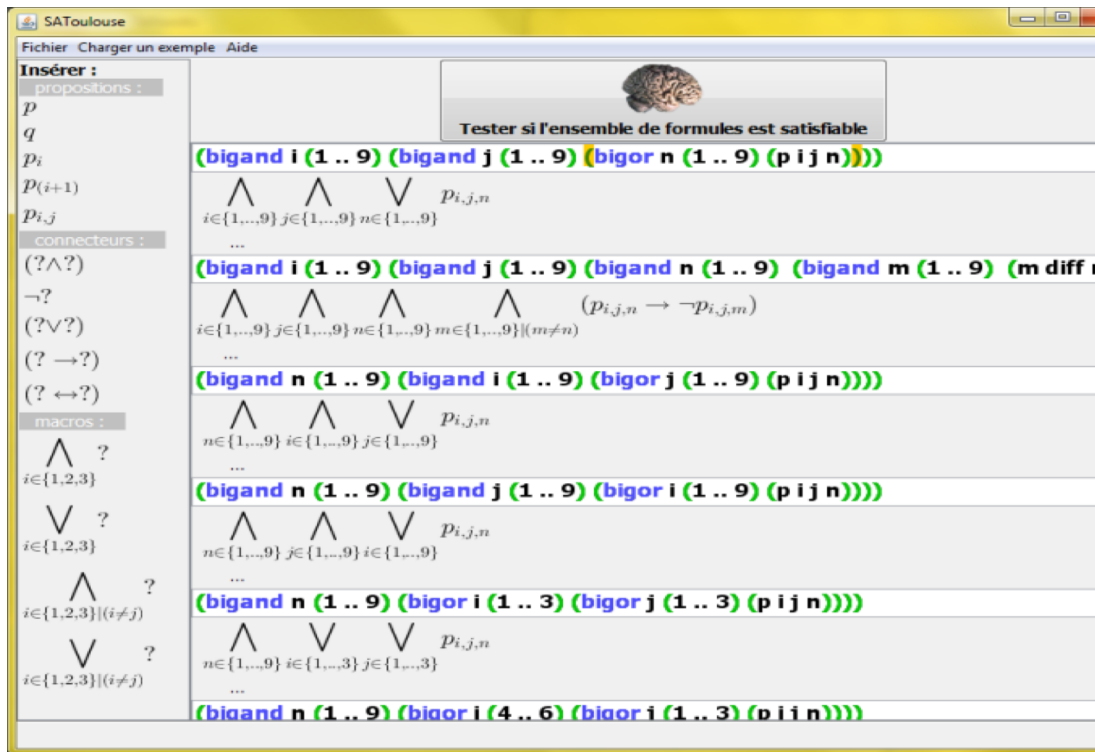
Ainsi, les problèmes traités sont formalisés en logique propositionnelle et ramenés au problème de « satisfiabilité ».

Une fois un problème décrit par un ensemble de formules booléennes, le logiciel permet de vérifier si leur intégralité est recevable (*satisfiable*) ; en d'autres termes, de savoir si le problème possède une solution.

Dans le cas où le problème a une solution, le logiciel donne une « valuation » (aussi appelée « interprétation ») : la « valuation » correspond au résultat du problème.

Le logiciel permet de remplir les champs de formules avec un chargement d'un fichier texte contenant des formules écrites ligne par ligne.

L'ensemble des modèles des connecteurs valides sont disponibles dans une palette à gauche et peuvent être chargés en cliquant dessus.



Capture d'écran du logiciel SAToulouse

B) Intérêts pour notre projet

Le projet est censé comme le rappelle le titre modéliser des débats logiques. Il était alors quasi-indispensable une fois ayant eu connaissance et eu à notre disposition le code de SAToulouse de l'utiliser pour résoudre la première grosse problématique à savoir comment on allait pouvoir vérifier la redondance, vérifier l'intégrité des formules, la consistance entre plusieurs d'entre elles etc.

Je me suis souvenu d'avoir utilisé en cours de Logique durant la deuxième année de licence ce dernier pour résoudre des sudokus. Je n'avais en revanche pas retenu que celui-ci était codé en JAVA et encore moins que sa licence était libre comme notre tuteur nous l'a rappelé.

Ne connaissant aucune bibliothèque particulière qu'elle soit de l'ordre de l'affichage, ou de l'ordre du calcul de satisfaisabilité, la tâche se serait annoncée particulièrement ardue voire très compromise.

La lecture et la compréhension de ce code source aura donc été la première longue séquence de travail. Et au vu de toutes les bibliothèques déjà attachées au dossier source j'ai compris que de tout coder soi-même était voué à l'échec.

C) Intégration/adaptation

a) Besoins supplémentaires par rapport à Satoulouse

- ✓ extraction d'atomes à partir de formules
- ✓ ajout de formules ou atomes dans la palette à gauche
- ✓ bouton pour vider tous les champs

- ✓ bouton pour ajouter un champ de saisie de formule
- ✓ emplacement pour champ de saisie d'une conclusion (pour l'ArgumentWriter)
- ✓ verrouillage des champs de saisie de formule pré-remplies lors d'un replace

b) Éléments inutiles

- ✓ fenêtre de résultat
- ✓ fenêtre d'aide
- ✓ chronomètre
- ✓ fenêtre d'attente de résultat
- ✓ fenêtre pour charger un Sudoku
- ✓ menu
- ✓ fenêtre « à propos »
- ✓ gestion des macros (bigand, bigorr)

c) Réalisation

J'ai tout d'abord après avoir cerné les liens entre les classes essayé d'ébaucher une fenêtre assez simplifiée qui permette de saisir une seule formule.

J'ai enlevé les macros (bigands, bigorr) ainsi que le menu pour simplifier un maximum l'interface qui allait dans notre projet devenir de toute façon secondaire, appelé par la GUI pour certains coups (assert, argue et replace),

Pour rendre le paquet plus personnalisé et moins hostile à mon collègue qui m'avait laissé gérer cette partie, j'ai en plus de supprimer l'appel à certaines classes sans intérêt pour notre jeu, supprimé un bon nombre de classes (Chronomètre, WaitDialog, SATResults, HelpPanel, DialogHelp, DialogGrille2D, SAToulouseAboutBox).

Pour SATResult bien que supprimé, j'ai repris le comportement pour avertir de la non consistance d'une formule et d'un argument par la suite.

Sat fonctionnait avec le duo de classes suivant : une extension de SingleFrameApplication lance une SingleView.

En cours de programmation événementielle, nous avons uniquement utilisé la bibliothèque JavaSwing donc pour éviter de rencontrer des problèmes par la suite et pensant qu'il était moins obsolète de fonctionner ainsi, j'ai essayé d'adapter en supprimant la SingleFrameApplication (sorte de main qui charge des ressources) et en faisant passer l'interface d'application (classe FrameView) indépendante à une JFrame.

J'ai pour cela repris les composants de la View un à un et j'ai recrée une JFrame.

Ceci dit la conversion posa un problème dans la mesure où certains paramètres étaient enregistrés dans la ressource du SingleFrameApplication qui devait disparaître donc j'ai repris les paramètres des fichiers ,properties pour les prendre en compte via Matisse(lien dans le glossaire).

Cette JFrame allait passer en JDialog un peu plus tard quand les événements seraient gérés, mais rester en JFrame avait l'intérêt de pouvoir la lancer directement et gagner du temps précieux lors des tests.

J'ai personnalisé le comportement du bouton validation.

Il me fallait un outil qui puisse:

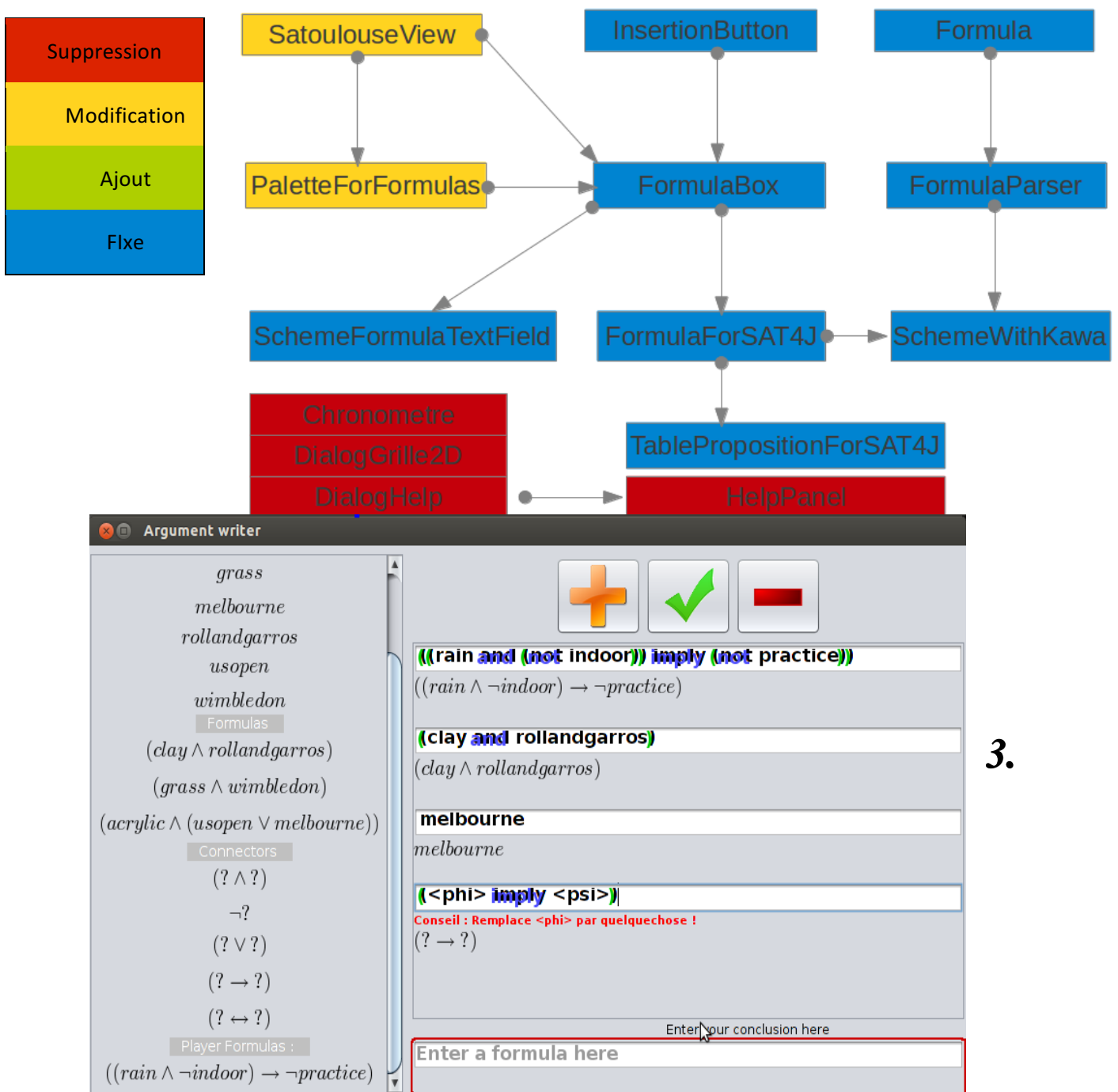
- ✓ charger les formules saisies dans la fenêtre courante des deux solvers

- ✓ ajoute dans le premier l'ensemble des formules de l'arbitre
- ✓ ajoute dans le second celles déjà évoquées par le joueur courant
- ✓ soumettre les deux à des tests de consistance
- ✓ avertir le joueur en cas d'erreur (selfconsistency et globalredundancy).

Une fois que ceci fonctionnait, j'ai voulu ajouter sur la palette flottante à gauche une aide à la saisie.

Pour extraire les atomes, j'ai modifié la classe PaletteForFormula et créer une fonction dans la classe SATSolverSAT4J (sur le modèle de la méthode déjà existante qui retournait les modèles de satisfaisabilité pour un problème).

d) Diagramme de classes



Justifications techniques

A) Sauvegarde/Restauration

a) Sauvegarder

On utilise les fonctions prédéfinies de `satoulouse.TextFile` permettant de supprimer un fichier en mémoire, d'écrire une liste ou une ligne dans un fichier.

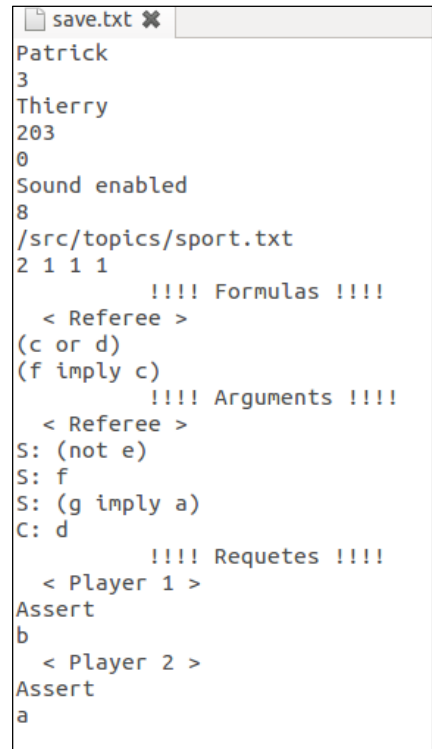
Le chemin aura été choisi via une fonction cette fois ci codée par nous même en utilisant un `JFileChooser`.

J'ai modifié les fonctions d'écriture de façon à ce que l'on écrive à la suite et non écraser à chaque nouvelle écriture.

Format type de sauvegarde

<nom du premier joueur: String>
<temps restant de parole
du premier joueur: int>
<activation du son: String>
<compteur de coups acceptés: int>
<thèmes (peut être vide)>
<nombre de formules de l'arbitre,
nombre d'arguments de l'arbitre,
nombre de requêtes en cours du joueur1,
nombre de requêtes en cours du joueur2>
<Formules de l'arbitre>
<Arguments de l'arbitre>
<Requêtes du joueur 1>
<Requêtes du joueur 2>

Exemple



```
save.txt ✖
Patrick
3
Thierry
203
0
Sound enabled
8
/src/topics/sport.txt
2 1 1 1
      !!!! Formulas !!!!
    < Referee >
(c or d)
(f imply c)
      !!!! Arguments !!!!
    < Referee >
S: (not e)
S: f
S: (g imply a)
C: d
      !!!! Requetes !!!!
    < Player 1 >
Assert
b
    < Player 2 >
Assert
a
```

Au départ je comptais enregistrer es formules et arguments des joueurs mais dans la mesure où chaque argument chez l'un correspond à une requête en cours Argue chez l'autre et où chaque formule correspond à un Assert chez l'autre, nous n'enregistrons en ce qui concerne les joueurs, uniquement les requêtes.

b) Restaurer

Je vais lire dans le fichier de sauvegarde sélectionné via la fonction de lecture également présente dans `satoulouse.TextFile` et je récupère une liste de String.

Cette liste sera séparée de son premier élément tout au long de la restauration et sera vide sauf erreur à la fin de celle ci.

B) Affichage des informations

a) Formules

Nous avons du utiliser une astuce pour afficher les propositions logiques telles qu'on les aperçoit dans la fenêtre de saisie (symbole \wedge pour and, \rightarrow pour imply etc),

Or pour afficher en Latex nous ne pouvions faire passer de simple String à nos listes d'affichage, il nous fallait convertir ces chaînes en Latex et installer se résultat sous forme d'ImageIcon dans le Label.

On a donc commencé par créer une classe FormulaLabel étendant la classe JLabel qui prend en paramètre un String ch (correspondant à l'écriture d'une formule) et qui effectue le protocole mentionné juste au dessus.

On a en second lieu créé notre propre rendu de cellule implémentant ListCellRenderer

La méthode getListCellRendererComponent nous retourne donc un FormulaLabel à qui on a envoyé le value casté en String (correspond à ch) paramétré de manière à ce qu'il change de coloris après une sélection.

On installe ce rendu à l'instanciation de la liste de formules dans le initComponents des panels du joueur et de l'arbitre.

b)

L'affichage des arguments posait rappel, un argument comprend logique) et un support qui est là support est une liste de priori le nombre de proposition

La difficulté était de pouvoir

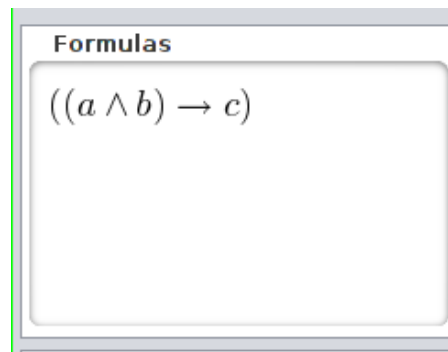
toutes ces informations sur l'interface graphique sans pour autant la surcharger.

C'est de la réflexion pour résoudre cette problématique que nous est venue l'idée d'afficher les arguments sous forme d'arbre. L'information principale (la conclusion) étant affichée par défaut, mais l'utilisateur a quand même accès aux autres informations (le support) par un simple clic pour déplier l'arbre.

Pour la réalisation, nous avons donc utilisé un élément Jtree qui permet l'affichage sous la forme d'arbre. Ensuite nous avons séparé le problème en deux : peupler l'arbre avec les arguments et afficher les arguments à l'écran.

Pour peupler l'arbre, la seule solution que nous avons trouvé est de réécrire entièrement le Jtree à chaque saisie d'argument par un joueur. La routine étant de prendre chaque argument du joueur (stocké dans l'objet player sous forme de liste), d'ajouter la formule correspondant à la conclusion comme nœud de l'arbre et d'attacher chaque formule du support comme feuille de ce nœud.

Une fois l'arbre construit, nous avons créé une classe getTreeCellRendererForArgument qui permet (en reprenant ce que nous avons fait pour les formules) d'afficher chaque formule sous sa forme Latex.

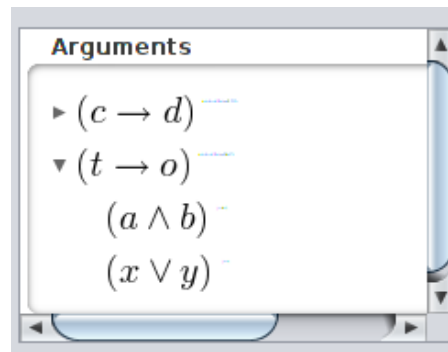


Arguments

des problèmes différents. Pour une conclusion (une proposition pour prouver la conclusion. Le proposition logique, sans savoir à qu'il peut y avoir.

donner l'accès à l'utilisateur à

c)



Requêtes

Les requêtes correspondent à une action effectuée par le joueur adverse. Par exemple si un joueur énonce un nouvel argument, celui ci sera présent sous la forme de requête pour le joueur adverse. A charge à lui ensuite de valider cet argument, de la contester, de demander à le compléter, etc...

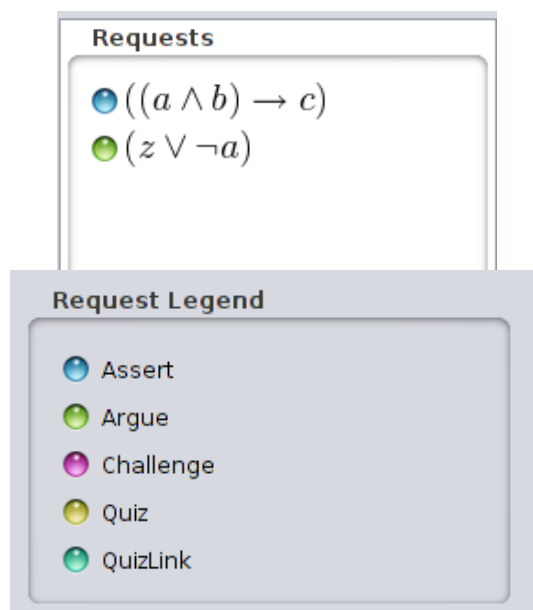
L'affichage de ces requêtes est passé par plusieurs stades. Au tout début lorsque peu de coups étaient encore codés (uniquement la proposition de formule « assert ») le réflexe que nous avons eu était d'afficher la formule correspondant à la requête. Mais avec les autres coups, plusieurs questions se sont posées. Est-ce que, pour une requête correspondant à une argument, l'on devait afficher l'argument sous forme d'arbre ? Comment différencier l'affichage d'une requête d'un argument simple et une requête de la demande de complétion d'un argument (« Challenge »)

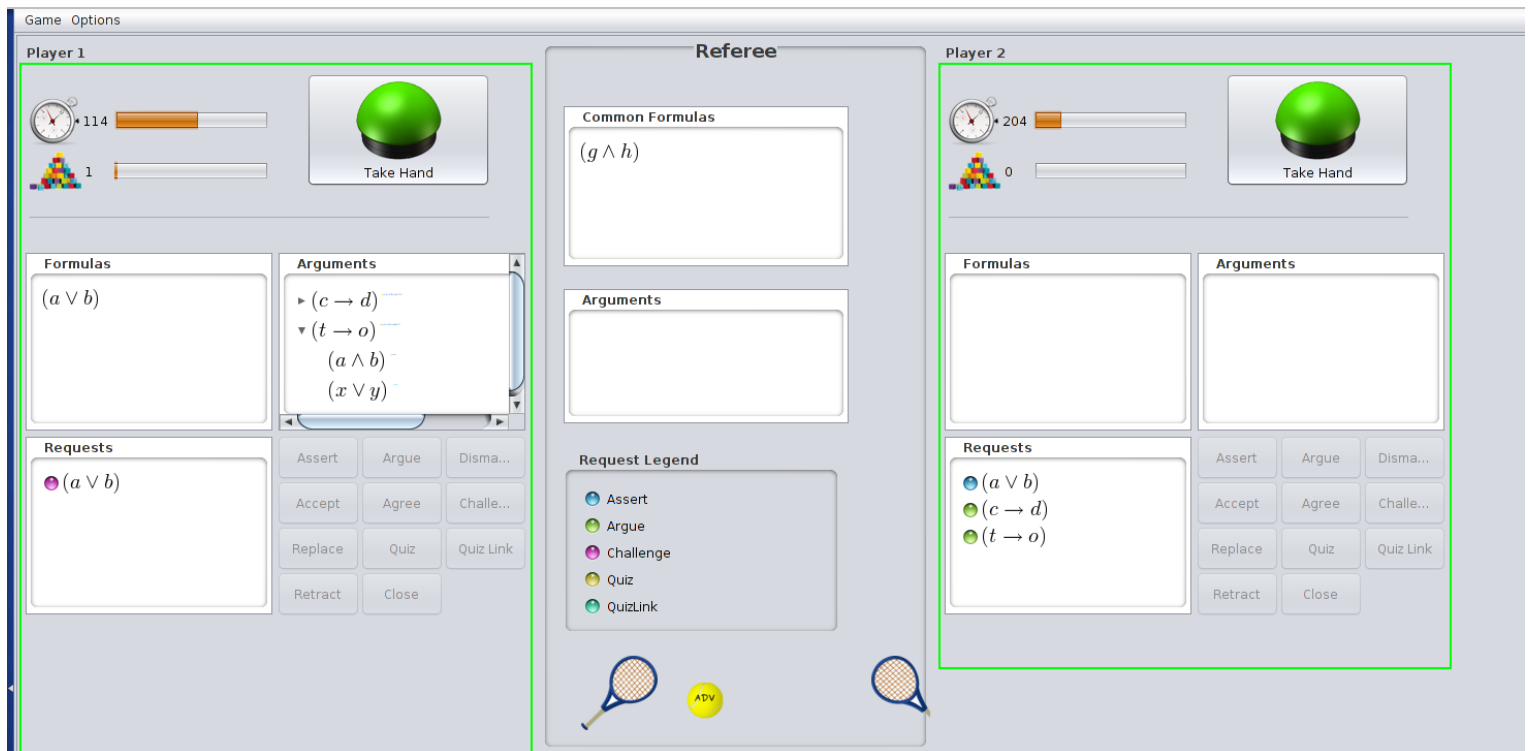
Nous avons donc, pour remédier à la situation un peu confuse que pourrait être l'affichage des requêtes, décidé d'utiliser un code couleur. Chaque couleur correspondant au type de requête, identifié par une puce devant la formule. Et pour ajouter de la clarté chez l'utilisateur une seule formule sera affichée (la conclusion pour un argument par exemple).

Cependant un problème conséquent pour l'affichage s'est posé. En effet, pour afficher une formule sous sa forme Latex dans une liste, c'est une image qui est utilisée (ImageIcon) or l'affichage de la puce de couleur requiert également l'affichage d'une image. Et il est impossible de créer un JLabel avec deux icônes.

Pour cela nous avons donc du créer une nouvelle classe CompoundIcon qui permet de fusionner deux images en une seule. Ici on fusionnera la puce de la bonne couleur avec à sa gauche l'image de la formule en Latex correspondante. Et c'est cette image fusionnée qui sera affichée dans la liste des requête

Techniquement, c'est dans la classe ListCellRenderForRequest que cette opération est effectuée. Ce renderer prend chaque requête de la Jlist, détermine le type de requête et le code couleur associé, réalise la fusion de la formule et de la puce et fait l'affichage.





C) Option pour lancement rapide d'une partie

Nous avons dès le départ adapté notre code de manière à ce qu'un simple clic sur un bouton dans le menu nous lance automatiquement une partie avec des paramètres configurés par défaut :

- ✓ noms de joueurs : Player 1 et Player 2
- ✓ temps de paroles à 250 secondes
- ✓ son activé
- ✓ aucun thème chargé
- ✓ personne n'a encore la main

Le procédure normale consistait à inscrire chaque joueur en cliquant sur un bouton et à entrer un pseudonyme non vide puis cliquer sur NewGame de l'arbitre.

Cela faisait un nombre trop important d'étapes pour effectuer des tests fréquents.

Ceci peut paraître inutile une fois que la sauvegarde et restauration sont parfaitement fonctionnels dans la mesure où il est possible de charger une partie rédigée en suivant les normes par sois même dans un éditeur de texte avec une configuration initiale.

Mais on s'est attaché à la restauration/sauvegarde assez tard du moins pas avant que le reste ne fonctionne.

D) Difficultés rencontrées

a) Adaptation du travail en binôme

Au tout début du projet, nous voulions explorer deux pistes. Construire notre logiciel en partant de zéro, ou utiliser SATOULOUSE comme base et partir de ce point là. J'étais chargé de trouver des pistes pour le démarrage « From Scratch » et David était chargé de regarder la réutilisabilité de SAToulouse.

David a progressé très rapidement et au bout de quelques jours, il a semblé plus facile de choisir la voie qu'il avait prise (plus facile et rapide vu les délais mais pas la plus adaptée ou maîtrisable). Cependant l'échange du projet a été compliqué. N'ayant pas forcément le même système ainsi que la même version de NetBeans, j'ai passé beaucoup de temps dans ce TER à corriger des choses pour arriver à faire fonctionner ce que David m'envoyait. Et pendant ce temps là, il avançait. Donc quand j'avais une version viable ou je pouvais travailler dessus, il me renvoyait une nouvelle version et ainsi de suite.

J'ai essayé de mettre en place un système de versionning pour palier à ce problème et pouvoir avancer tous les deux mais je n'ai pas réussi à la faire adapter à David. (<http://code.google.com/p/debate-sat-ter/>)

b) Affichage de plusieurs icônes dans un Jlabel

L'affichage d'une formule avec une puce de couleur pour les requêtes demandait à faire cohabiter deux images dans le même Jlabel. Beaucoup de recherches ont été nécessaire pour trouver une façon adéquate de faire ça.

La façon retenu a été d'utiliser des fonctions permettant de créer une image à partir de deux autres et d'utiliser cette image créée « à la volée » pour le Jlabel.

1. Satisfactions

Nous sommes dans l'ensemble assez satisfaits du produit que nous avons conçu.

Nous avons pu instaurer des éléments que nous n'avions pas défini à la spécification et à la définition des besoins.

- ✓ affichage des arguments sous forme d'arbre déroulant
- ✓ association de chaque type de requête avec sa couleur (précisé dans une légende)
- ✓ affichage de l'évolution du score
- ✓ son

2. Regrets

Ceci dit nous n'avons pas recouvert l'ensemble des besoins que l'on avait évoqué lors des spécifications.

A) L'intelligence artificielle

Plusieurs raisons peuvent expliquer cela, on était assez juste en délai, le code n'est pas parfaitement adapté pour intégrer une IA rapidement.

Sans compter qu'elle aurait été trop perfectible:

- ✓ la génération aléatoire de formules étant particulièrement difficile, la génération de formules concises et cohérentes l'est d'autant plus.
- ✓ difficulté pour définir une stratégie d'argumentation ou de réplique
- ✓ besoin d'une fonction « decode » qui à partir d'un support calcule si la conclusion peut être prouvée.

L'intégration d'une telle fonction aurait été plus facile si elle avait été prise en considération dès le début du projet. Avec une réflexion plus profonde sur la structure du programme, le gestion des coups... Et surtout une réflexion sur la valuation des coups qui aurait pu servir à la fois pour établir le score des joueurs et pour donner à l'IA un repère (choisir les coups en fonction du meilleur score).

B) Le compte-rendu

Nous comptons au début enregistrer au fur et à mesure dans un fichier ce qui se passait en rendant le

dialogue parfaitement audible et naturel.

Par exemple pour un quizLink on aurait écrit « Player 2: Je ne vois pas le rapport avec le sujet du débat Mr » ou « Player 1: Je suis d'accord quand au fait que ... entraine ... ».

Mais le plus compliqué résidait dans la traduction de formules.

Nous n'avons pas trouvé de solutions fiables pour parser une formule, sûrement qu'une méthode récursive peut le permettre pour n'importe quelle profondeur de formule.

C) Application des méthodes de développement

David et moi n'avons pas choisi le même parcours pour ce second semestre de 3ème année. J'ai choisi ISI tandis que David a pris le parcours Informatique Fondamentale. J'ai appris beaucoup de choses au cours de ce semestre concernant le cycle de développement d'un logiciel et les méthodes que cela implique.

J'aurais aimé pouvoir mettre en place les différentes phases de développement pour produire le logiciel (Architecture, Codage, Phase de test etc..) Peut-être que les délais du TER ne se prêtaient pas à ce type de gestion. Mais le plus grand obstacle a été que David n'étant pas sensibilisé à ce type de méthode, il aurait été difficile de lui imposer une telle gestion.

D) Bonnes Conventions de codage

Je ne pense pas que le code rendu peut être réutilisable facilement. J'ai moi même eu des difficulté a me retrouver d'une version sur l'autre.

Ceci est principalement du au fait que le code n'est que très peu commenté. J'aurais aimé passer plus de temps à commenter le code que j'ai écrit de manière à faciliter sa compréhension et donc le travail de groupe.

V- Conclusion

Ce TER m'a tout d'abord permis de voir le fonctionnement de ce type de projet. Nécessitant une grande autonomie et une organisation particulière. Le sujet paraissait intéressant de prime abord, mais il a fallu débroussailler beaucoup de choses et se recentrer sur des fonctions primordiales.

L'utilisation de SAToulouse a été primordiale, surtout pour ce qui est du gain de temps. Beaucoup de problèmes ont été résolus, au détriment de la maîtrise du logiciel. Par exemple, un démarrage de zéro aurait permis de coder les atomes sous forme d'une classe comprenant une traduction littérale, un symbole et un ensemble de sujets liés. Ceci aurait été beaucoup plus souple pour la suite qu'une simple chaîne de caractère comme utilisé dans SAToulouse.

Pour la suite de ma formation, je souhaiterais m'orienter vers la filière IHM. Ce projet m'a permis de construire une interface graphique quasi-complète. J'ai souvent essayé de me positionner en tant qu'utilisateur pour pouvoir choisir les bonnes solutions d'interactions.

J'ai aussi eu l'occasion d'utiliser certains éléments multimédia avec un peu plus de profondeur telle que la gestion ou la construction d'images ou l'utilisation de sons. J'aurais aimé approfondir un peu plus cet aspect comme par exemple utiliser des threads pour le son au lieu de bloquer l'interface.

J'ai également le sentiment d'avoir progressé en programmation JAVA pure mais je suis un déçu de ne pas avoir pu appliquer ce que j'avais appris lors de ce semestre en terme de méthode de développement. Peut être que la forme du stage (TER) ne s'y prêtait pas. Et cela m'a donc donné envie de découvrir l'autre type de stage (Entreprise) pour pouvoir me donner des éléments de comparaison. Ceci afin de faire fructifier ses expériences et me permettre de faire les bons choix pour mon cursus et mon avenir professionnel.

VI- Glossaire

GUI : Graohical User Interface

Prolog : Langage de programmation logique

JFrame : Classe Java de la bibliothèque SWING pour gérer une fenêtre

setVisible : méthode permettant d'afficher un élément graphique (SWING)

JDialog : fenêtre pré-configurées permettant de communiquer simplement avec l'utilisateur

VII- Annexe

✓ Notice du jeu