Université Pierre et Marie Curie

Projet PSAR
Une interface graphique pour la logique

Auteurs :
Bastien RIGAULT
Sandra LADURANTI

Référents :
Béatrice BERARD
Mathieu JAUME
Bénédicte LEGASTELOIS

29 mars 2016

Résumé

L'objectif de ce projet, réalisé dans le cadre de l'UE PSAR (4I408) est de concevoir un logiciel pédagogique pour l'apprentissage de la logique du premier ordre. Ce logiciel apportera notamment un support visuel sous la forme d'un jardin et de fleurs permettant d'appréhender plus facilement les formules de la logique. Les étudiants auront à leur disposition deux fenêtres : l'une permettant de concevoir un jardin en y positionnant des fleurs et l'autre permettant d'écrire des formules à l'aide d'un clavier visuel. Ils pourront alors soit construire un jardin respectant un ensemble de formule données, ou au contraire établir des formules à partir d'un jardin donné. Les formules pourront être analysées de manière automatique afin de vérifier leur cohérence. Enfin, les étudiants pourront sauvegarder leurs jardins et leurs formules pour les réutiliser ultérieurement.
La première partie de ce cahier des charges présente d'abord succinctement la syntaxe de la logique du premier ordre mise en jeu. La section suivante définit les fonctionnalités principales du logiciel ainsi que les outils nécessaires à sa réalisation. Enfin les deux dernières parties concernent l'organisation et la répartition du travail ainsi que des éventuelles fonctionnalités optionnelles.

Table des matières

Chapitre 1

Présentation du projet

1.1 Sujet

Le sujet ne peut être abordé sans un rapide point sur les différents aspects de la logique du premier ordre telle qu'elle sera utilisée dans le projet :

1.1.1 Introduction à la logique du premier ordre

Une formule de la logique des prédicats telle qu'elle sera écrite par les étudiants est constituée de termes, de prédicats, de quantificateurs et de connecteurs logiques. Prenons par exemple la formule suivante :

$$\forall x Rose(x) \Rightarrow estRouge(x)$$

Pour écrire une telle formule, l'étudiant utilise des termes comme briques de base. Ces termes correspondent soit à l'ensemble $F_0 = \{a, b, ..., t\}$ des constantes, soit à l'ensemble $X = \{v, w, x, y, z\}$ des variables.
Ensuite, il dispose également d'un ensemble de prédicat P_i d'arité i, pour $i \in \{1, 2, 3\}$. Chaque prédicat de P_1, P_2, P_3 prend ainsi respectivement 1, 2 ou 3 termes en argument. Dans notre exemple, $Rose(x)$ et $estRouge(x)$ sont deux prédicats de P_1 et prennent la variable x en argument.
À partir de ces formules de base, l'ensemble des formules est défini inductivement : si φ est une formule alors $\neg\varphi$ est également une formule. De même si φ_1 et φ_2 sont des formules alors $\varphi_1 \wedge \varphi_2$, $\varphi_1 \vee \varphi_2$ et $\varphi_1 \Rightarrow \varphi_2$ aussi. Enfin, si $x \in X$ et φ est une formule alors $\forall x\varphi$ et $\exists x\varphi$ aussi.
Ce projet utilise un ensemble de prédicat prédéfinis que l'on regroupera par arité :

Unaire définissant l'état d'une fleur, elle peut être :
- une rose, une paquette ou une tulipe
- petite, moyenne ou grande
- rouge, rose ou blanche
- à l'est, à l'ouest, au sud ou au nord

Binaire définissant une relation entre 2 fleurs, la fleur a peut être :
- à l'est, à l'ouest, au sud ou au nord de la fleur b
- à la même latitude ou longitude que la fleur b
- plus grande, plus petite ou de même taille que la fleur b
- de la même couleur que la fleur b

Ternaire définissant une relation entre 3 fleurs, la fleur c peut être :
- entre la fleur a et la fleur b

La section suivante présente quelques exemples de formules utilisant différents prédicats.

1.1.2 Exemple

On peut par exemple écrire les huit formules ci-dessous.

(f1) d est une rose : Rose(d)
(f2) Toutes les fleurs sont des roses : $\forall x Rose(x)$
(f3) Il existe une rose : $\exists x Rose(x)$
(f4) Toute fleur blanche est plus petite qu'au moins une fleur située à son est :
$\forall x(est\_blanc(x) \Rightarrow \exists y(plus\_petit\_que(x,y) \wedge a\_l\_est\_de(y,x)))$
(f5) Toute fleur est à l'est, ou à l'ouest, ou au sud, ou au nord :
$\forall x(a\_l\_est(x) \vee a\_l\_ouest(x) \vee au\_sud(x) \vee au\_nord(x))$
(f6) Toutes les grandes fleurs sont rouges et il n'existe pas de fleur blanche au sud d'une fleur rouge :
$\forall x(est\_grand(x) \Rightarrow est\_rouge(x)) \wedge \neg\exists x(est\_blanc(x) \wedge \exists y(est\_rouge(y) \wedge au\_sud\_de(x,y)))$
(f7) Il existe une fleur rouge au nord de la fleur g : $\exists x(est\_rouge(x) \wedge au\_nord\_de(x,g))$
(f8) Il existe une unique rose rouge :
$\exists x((Rose(x) \wedge est\_rouge(x)) \wedge (\forall y(Rose(y) \wedge est\_rouge(y)) \Rightarrow x \stackrel{\circ}{=} y))$

Les exemples ci-dessus seront donc appliqués dans un jardin tel que présenté dans la section suivante.

1.1.3 Application de la logique dans le jardin

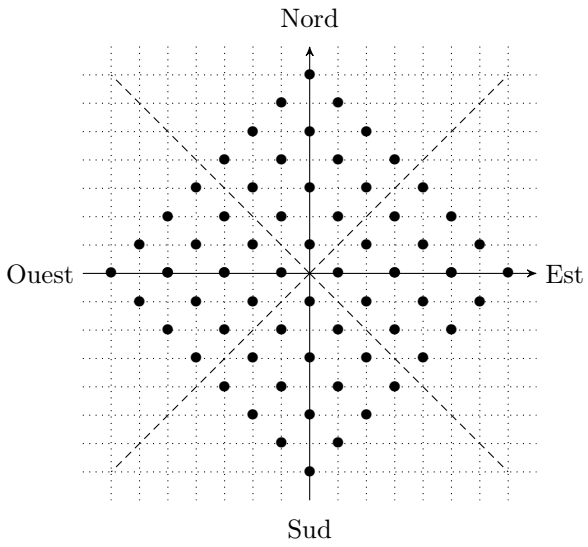

FIGURE 1.1 – Places d'un jardin

Les termes considérés dans le projet désignent des fleurs positionnées sur une grille. La figure 1 représente la grille : les points noirs symbolisent les places où peuvent être disposées les fleurs (il y a au plus une fleur par place). Chaque place est identifiée par ses coordonnées (le point $(0,0)$ est au centre de la grille). L'ensemble des places possibles est donc :

$$\text{Places} = \left\{ \begin{array}{c} (0,7), \\ (-1,6),(1,6), \\ (-2,5),(0,5),(2,5), \\ (-3,4),(-1,4),(1,4),(3,4), \\ (-4,3),(-2,3),(0,3),(2,3),(4,3), \\ (-5,2),(-3,2),(-1,2),(1,2),(3,2),(5,2), \\ (-6,1),(-4,1),(-2,1),(0,1),(2,1),(4,1),(6,1), \\ (-7,0),(-5,0),(-3,0),(-1,0),(1,0),(3,0),(5,0),(7,0), \\ (-6,-1),(-4,-1),(-2,-1),(0,-1),(2,-1),(4,-1),(6,-1), \\ (-5,-2),(-3,-2),(-1,-2),(1,-2),(3,-2),(5,-2), \\ (-4,-3),(-2,-3),(0,-3),(2,-3),(4,-3), \\ (-3,-4),(-1,-4),(1,-4),(3,-4), \\ (-2,-5),(0,-5),(2,-5), \\ (-1,-6),(1,-6), \\ (0,-7) \end{array} \right\}$$

Chaque fleur appartient à une espèce (les roses, les pâquerettes et les tulipes), est d'une certaine taille (grande, moyenne ou petite) et d'une certaine couleur (rouge, rose, blanche).

$$\begin{aligned} Especes &= \{rose, paquerette, tulipe\}, \\ Tailles &= \{grand, moyen, petit\}, \\ Couleurs &= \{rouge, rose, blanc\}. \end{aligned}$$

Certaines fleurs ont un nom : il s'agit d'une constante de F_0 (deux fleurs différentes ne peuvent pas avoir le même nom). Un jardin j est la donnée d'une grille sur laquelle sont disposées des fleurs (chaque jardin contient au moins une fleur) et est représentée par une liste de quintuplets. Chaque quintuplet $((x,y),e,t,c,n) \in j$ est un élément du produit cartésien :

$$Places \times Especes \times Tailles \times Couleurs \times (F_0 \cup \{None\})$$

et exprime qu'une fleur de nom n (n = None si la fleur n'a pas de nom), d'espèce e, de taille t et de couleur c se trouve à la place (x,y) dans le jardin j. L'ensemble des jardins possibles est donc :

$$J = \wp(Places \times Especes \times Tailles \times Couleurs \times (F_0 \cup \{None\}) \setminus \{\emptyset\}$$

1.2 Problématiques soulevées

Deux grosses problématiques sont soulevées par la réalisation de ce projet.
— La récupération et vérification de l'intégrité syntaxique des formules entrées dans le programme avant de les envoyer au script de vérification de leur véracité dans un jardin donné ;
— La communication entre deux langages différents et une interface graphique.

1.3 Hypothèse de solution

1.3.1 Problématique de l'intégrité syntaxique

La solution envisagée est l'utilisation d'un générateur d'analyseur syntaxique, idéalement via l'outil Grammatica version 1.6 (libre de droit sous licence BSD). Après que l'outil ait reçu une grammaire spécifique en entrée, il produit un parseur pour cette grammaire en C#. Ce code n'est généré qu'une seule fois et est capable d'analyser des formules données sous forme de chaînes de caractères. Le résultat final obtenu est soit un message d'erreur détaillé si la formule est incorrecte soit un arbre syntaxique représenté en C# comme l'illustre la figure suivante :

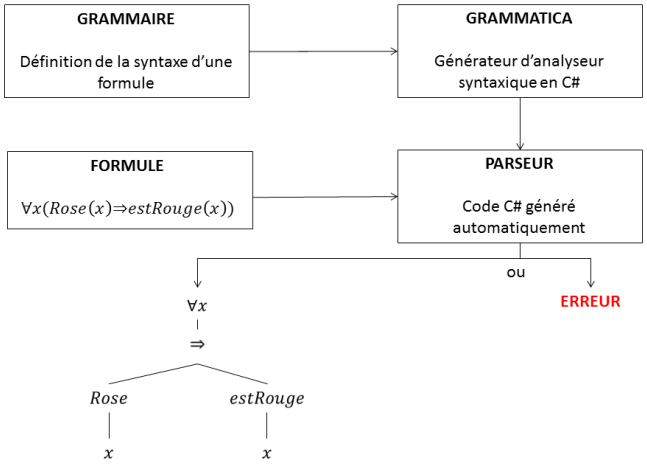

FIGURE 1.2 – Arbre Syntaxique

1.3.2 Problématique de la communication

Le sujet du projet est fourni avec un script de vérification des formules. Il est donc important de pouvoir faire communiquer ce script avec le langage de développement choisi : C#. !!!!!!!!TODO : voir si il faut parler direct du python, expliquer pourquoi on est passé sur python ou parler d'ocaml ? !!!!!!!!

Chapitre 2

Analyse de l'existant

Le projet tourne autour d'un script préalablement fourni en python.

2.1 Script python

Le script permet de créer un jardin en disposant les différents éléments avec leurs attributs sur diverses coordonnées. En premier lieu il est nécessaire de bien créer un jardin, il serait inutile de tester une formule sans contexte autour. Le script permet ensuite de créer une formule, celle-ci doit être préalablement vérifiée pour que sa syntaxe ne contienne aucun erreur. Dans un dernier temps le script analyse la formule dans le contexte du jardin donné et retourne un booléen en résultat.

2.2 Bilan récapitulatif

Voici le tableau (cf. fig. 2.1) récapitulatif de l'analyse de l'existant :

Solution	Création	Vérification	Analyse
Jardin	Oui	Non	Non
Formule	Oui	Non	Oui

FIGURE 2.1 – Tableau récapitulatif des solutions

Chapitre 3

Analyse des besoins

Le projet s'axe sur le besoin fonctionnel de base de l'interface graphique autour duquel gravitent des besoins non fonctionnels mais nécessaires au bon fonctionnement de l'application.

3.1 Besoins fonctionnels

Après une analyse des besoins fonctionnels du projet, nous avons défini deux sous catégories. D'un côté, les besoins graphiques, de l'autre, les besoins liés à la syntaxe des formules.

3.1.1 Interface Graphique

L'interface graphique doit être ergonomique et user-friendly, le but est d'accompagner les étudiants dans l'apprentissage de la logique du premier ordre de la manière la plus agréable possible. Pour cela un certain nombre d'exigences en terme de GUI ont été mis en place :

- un jardin représenté par une grille de taille fixe ;
- des fleurs pouvant être disposées sur le jardin et ayant un visuel différent selon leurs espèces, tailles et couleurs ;
- un menu permettant de positionner de nouvelles fleurs à la souris et d'en changer les caractéristiques ;
- un menu permettant d'écrire et vérifier des formules de la logique du premier ordre à l'aide d'un clavier virtuel et d'une liste de 5 variables et de 20 constantes ;
- un menu permettant de sélectionner une variable ou une constante ;
- un clavier visuel permettant de sélectionner les connecteurs de la logique (not, et, ou, pour tout, etc.) ;
- un menu permettant de sauvegarder et de charger des jardins et/ou des ensembles de formules.

Aperçu du rendu souhaité :

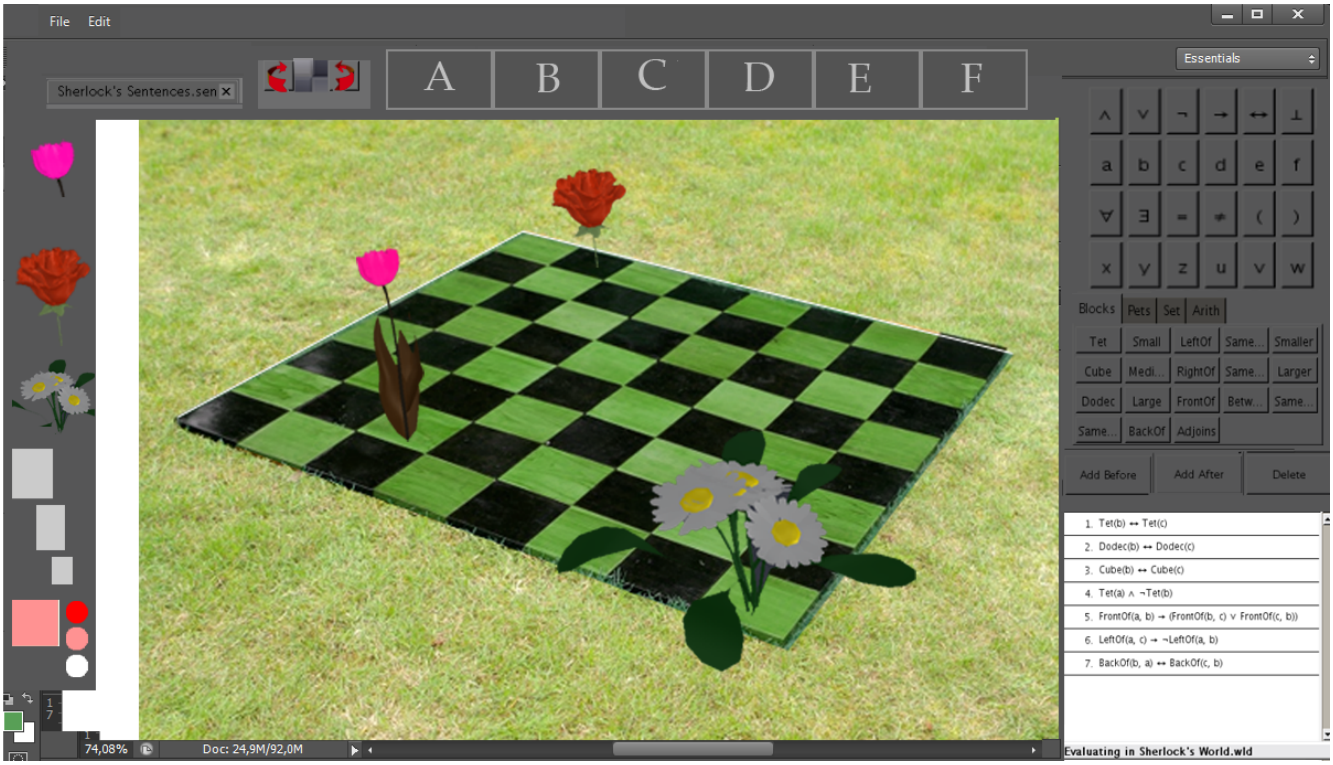

FIGURE 3.1 – Rendu attendu

3.1.2 Analyse Syntaxique

L'analyse syntaxique est un besoin lié directement au script fourni avec le sujet du projet. Si cette partie n'est pas parfaitement fonctionnelle, le script ne pourra donc pas fonctionner correctement et le projet ne pourra aboutir.
Idéalement l'analyse syntaxique pourra retourner le problème exact dans la formule entrée par l'utilisateur, cependant ce point n'est pas une nécessité, cette partie doit en priorité stopper le programme et notifier l'utilisateur du non respect de la syntaxe dans l'une de ses formules.

3.2 Besoins non-fonctionnels

Il n'est utile d'aborder qu'un seul besoin non-fonctionnel : la communication entre deux langages différents, ce point étant technique nous préférons revenir dessus plus tard et seulement l'évoquer ici. Il a cependant été pris en compte les contraintes de développement, détaillées à la fin de cette partie.

3.3 Développement

Le développement est réparti en tâches parallélisées afin d'optimiser le temps de réalisation.

3.3.1 Tâches

Le développement s'axe sur trois grandes tâches, la communication inter langages, l'analyse syntaxique et la création de l'interface graphique. Le but étant de relier les trois blocs ensemble pour obtenir l'application finale.

Priorité	Nom	Raison
1	Communication inter langages	Doit être vérifié en premier car sinon on ne pourra pas utiliser l'Analyseur Syntaxique
2	Analyseur Syntaxique	On doit pouvoir entrer n'importe quel formule et être capable de détecter au plus vite la moindre erreur dans celle-ci
3	Laison Communication-Analyseur	Comme les principales fonctionnalités permettant de tester sont opérationnelles, nous pouvons passer à cette tâche.
4	Création de l'interface graphique	Dernière fonctionnalité essentielle à mettre en place.
5	Animation des fleurs	Non-essentiel, mais apporterait un plus au projet.
6	Fonctionnalité de retour en arrière	Non-essentiel, mais apporterait un plus au projet.

FIGURE 3.2 – Tableau récapitulatif des tâches

3.3.2 Organisation du code

Dans le cas où l'analyseur syntaxique detecterait une erreur de formule, celui-ci ne retournera pas d'objets mais notifiera le bloc de communication de l'erreur. Ce dernier remontera donc le soucis au bloc GUI qui l'affichera à l'utilisateur.

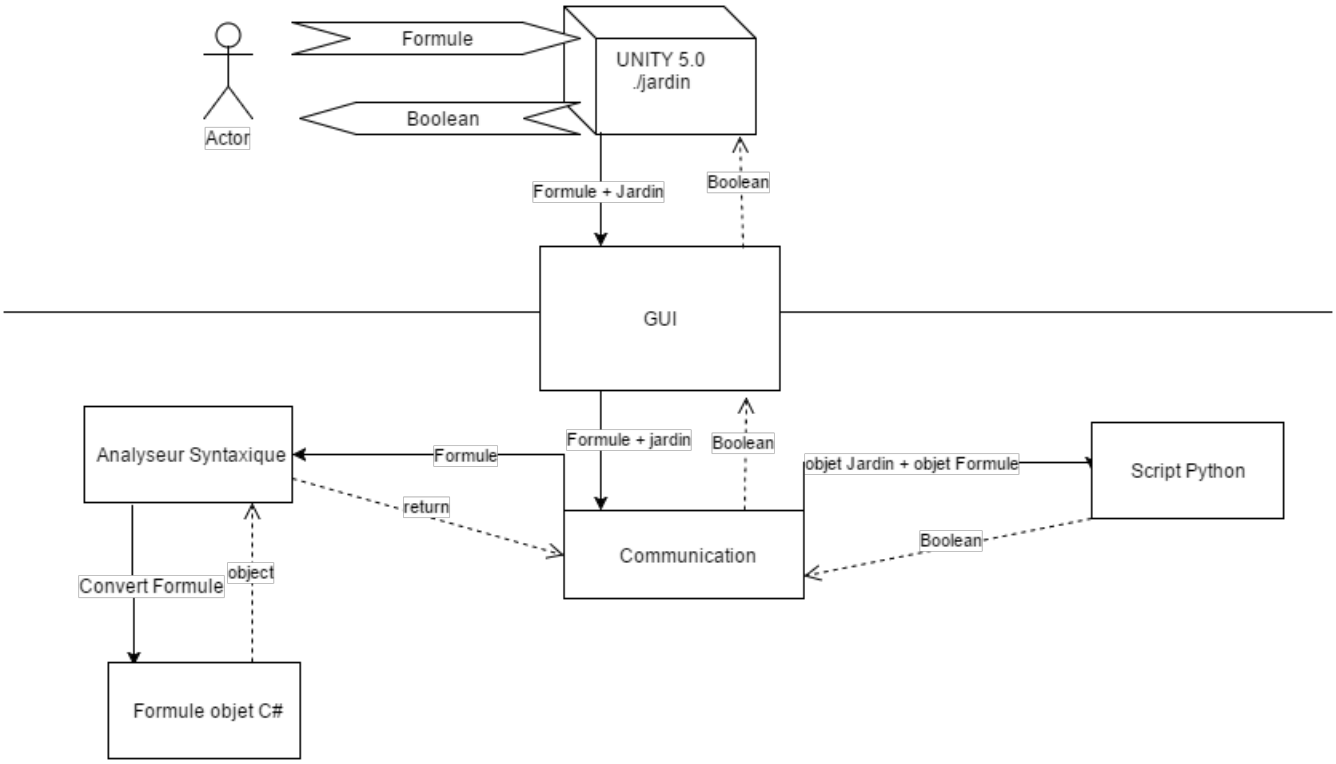

FIGURE 3.3 – organisation des blocs

Chapitre 4

Choix techniques

Dans cette partie nous cherchons à décrire dans un premier temps les divers choix techniques et spécificités du projets plus en détail.

4.1 Analyse Syntaxique

TODO : section à remplir

4.1.1 Sous-partie 1

4.1.2 Sous-partie 2

4.1.2.1 Sous-sous-partie 1

TODO : section à remplir par sandra

4.2 Communication

4.2.1 Problématique rencontrée

4.2.2 Sous-partie 2

Paragraphe 1 (n'apparaitra pas dans l'index)

Paragraphe 2 Bla

Paragraphe 3 Bla

4.2.3 Sous-partie 3

Bla

Chapitre 5

Résultats

5.1 Partie 1

Intro

5.1.1 Sous-partie 1

5.2 Partie 2

Intro

Sous-partie 1 (’apparaitra pas dans l’index)

Sous-partie 2

Chapitre 6

Bilan

Intro / Rappel Contexte
Nous avons donc pu en tirer la problématique suivante :

Problématique du sujet

Annexes

Annexe 1

Intro

1 Partie 1

1.1 Sous-partie 1

2 Partie 2

2.1 Sous-partie 1

Annexe 2

Intro

Prérequis

1 Partie 1

2 Partie 2

Attention !
Texte d'avertissement

3 Partie 3