UPMC, 4I408
Sandra Laduranti
Bastien Rigault

Année 2015–2016

Projet SAR
Une interface graphique pour la logique

Cahier des charges

1 Description du projet

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.

1.1 Logique du premier ordre

1.1.1 Les bases de la logique

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 formule utilisant différents prédicats.

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 :
$\exists x((Rose(x) \wedge est\_rouge(x)) \wedge (\forall y(Rose(y) \wedge est\_rouge(y)) \Rightarrow x \stackrel{\circ}{=} y))$

2 Fonctionnalités principales

2.1 Le jardin

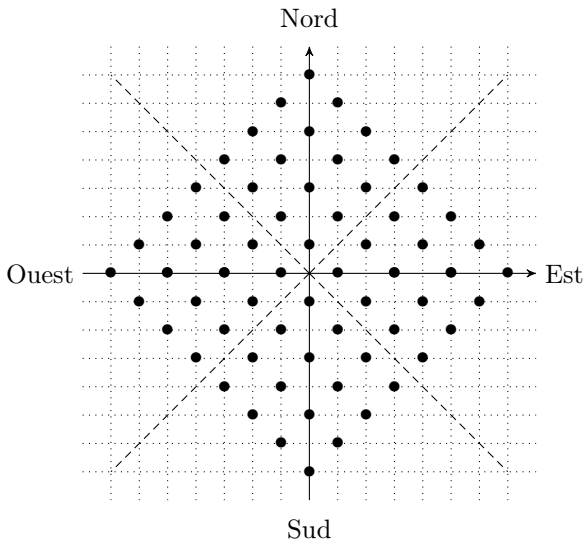


FIGURE 1 – Places d'un jardin

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).

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

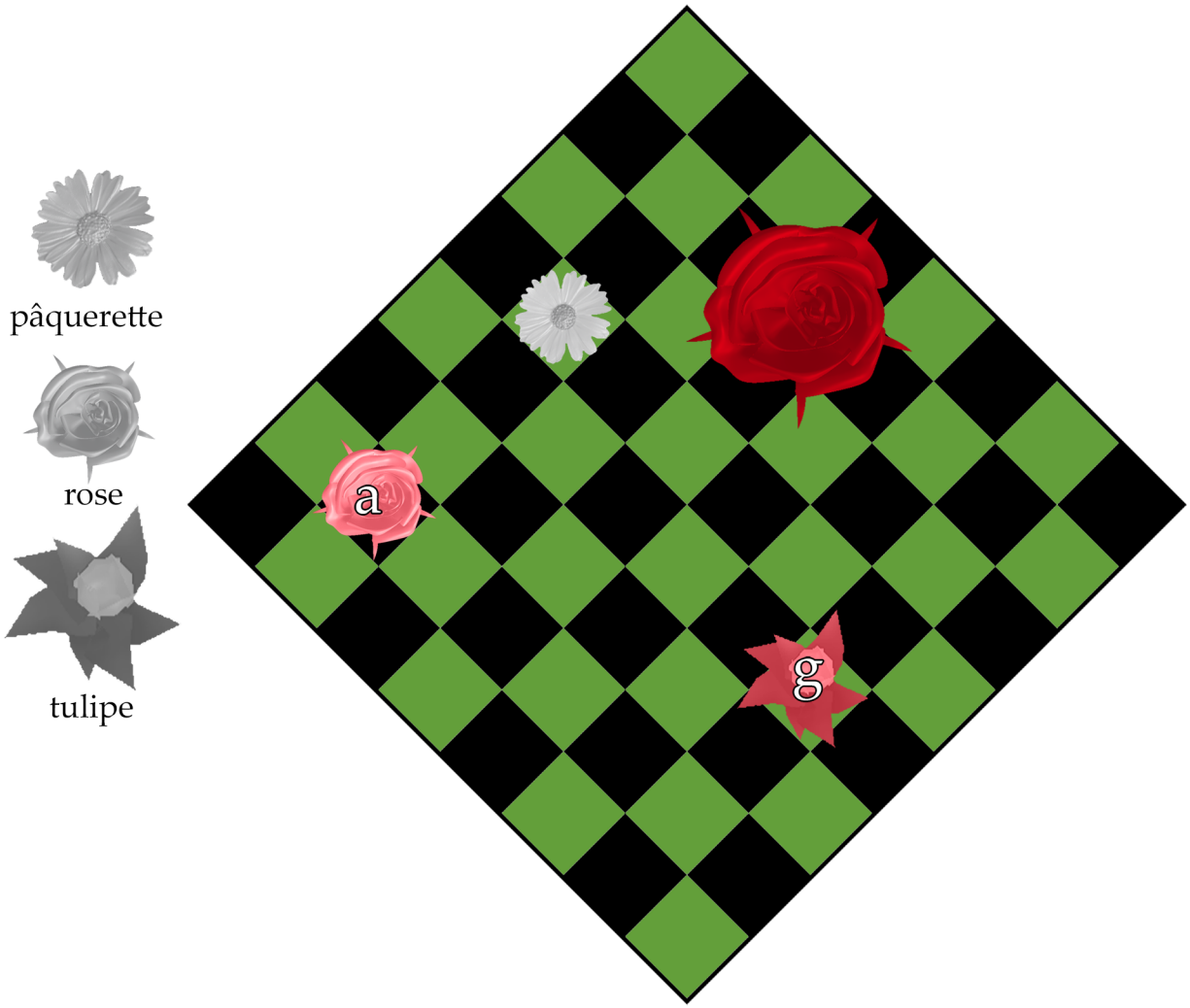
Certaines fleurs ont un nom : il s'agit d'une constante de F_0 (deux fleurs différentes ne peuvent pas avoir le même nom). Un jardin j est la donnée d'une grille sur laquelle sont disposées des fleurs (chaque jardin contient au moins une fleur) et est représentée par une liste de quintuplets. Chaque 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 :

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

2.2 Exemples de formules



;

Voici quelques exemples de formules dans le jardin ci-dessus :

(f1) : FAUX Toutes les fleurs sont des roses : $\forall x Rose(x)$

(f2) : VRAI Il existe une rose : $\exists x Rose(x)$

(f3) : VRAI Toute fleur blanche est plus petite qu'au moins une fleur située à son est :
$\forall x(est\_blanc(x) \Rightarrow \exists y(plus\_petit\_que(x,y) \wedge a\_l\_est\_de(y,x)))$

(f4) : FAUX Toutes les tulipes sont rouges : $\forall x Tulipe(x) \Rightarrow estRouge(x)$

(f5) : VRAI Toutes les grandes fleurs sont rouges et il n'existe pas de fleur blanche au sud d'une fleur rouge :
$\forall x(est\_grand(x) \Rightarrow est\_rouge(x)) \wedge \neg\exists x(est\_blanc(x) \wedge \exists y(est\_rouge(y) \wedge au\_sud\_de(x,y)))$

(f6) : FAUX a est une pâquerette : $Paquerette(a)$

(f7) : VRAI Il existe une fleur rouge au nord de la fleur g : $\exists x(est\_rouge(x) \wedge au\_nord\_de(x,g))$

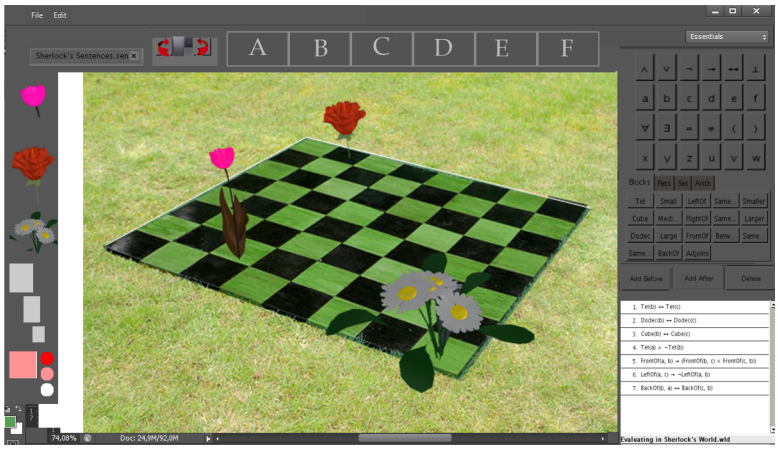
(f8) : VRAI Il existe une unique rose rouge :
$\exists x((Rose(x) \wedge estrouge(x)) \wedge (\forall y(Rose(y) \wedge est\_rouge(y)) \Rightarrow x \mathring{=} y))$

2.3 Aspect graphique

2.3.1 En résumé

Les éléments nécessaires sont :
— 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 l'aspect,
— 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.

2.3.2 Simulation de rendu

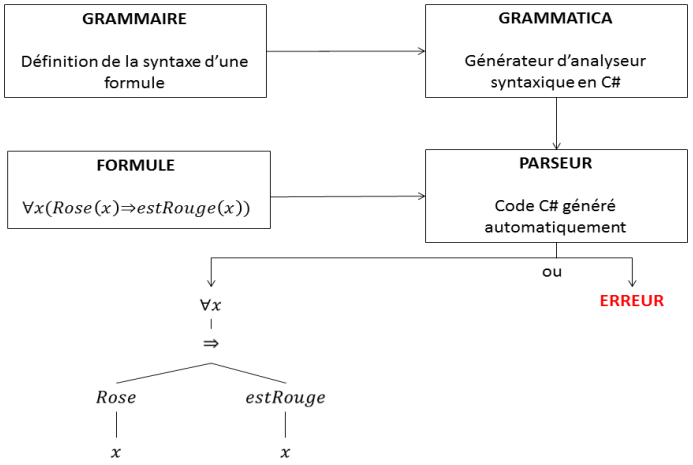


;

3 Aspects techniques

3.1 Analyse syntaxique

L'analyse syntaxique se fera à l'aide de l'outil Grammatica version 1.6 (libre de droit sous licence BSD). Il s'agit d'un générateur d'analyseur syntaxique qui, après avoir reçu une grammaire spécifique en entrée, 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 :



;

3.2 Vérification des formules

La vérification des formules sera réalisée en trois parties distinctes, à savoir leur récupération, la vérification de leur intégrité syntaxique (vue au chapitre précédent) et enfin la vérification de leur validité dans le jardin donné.

3.2.1 Choix des langages

Le choix des langages a été régi par la contrainte de communication entre deux langages de programmation différents. L'outil de vérification de formules fourni de base étant en Ocaml, le besoin premier était de trouver comment communiquer avec ce dernier. En parallèle il fallait également prendre en compte les contraintes esthétiques, l'application devant être au final à la fois solide et conviviale. La solution de la programmation en Java a donc été immédiatement éliminée par souci de manque d'esthétisme dans ses GUI.

Suite à de nombreuses recherches, le choix s'est donc porté sur du C#, offrant un outil de développement permettant de faire le pont entre Ocaml et C#. De plus le choix de C# a été confortée par la possibilité d'utiliser ce langage avec le moteur de jeu Unity 5. Les avantages de ce moteur étant tout d'abord sa gratuité mais également son moteur 3D qui entre en parfaite adéquation avec les objectifs finaux du projet.

3.2.2 Interprétation des formules

En premier lieu, l'étudiant entrera dans l'application, à l'aide de sa souris et sur le clavier optique fourni par le programme, une formule à valider. La formule sera alors récupérée via les événements de la GUI de Unity et seront transmis au module d'analyse syntaxique de l'application via C# (vu au chapitre de l'analyse syntaxique). En supposant que la formule ait passé la vérification syntaxique, elle sera alors envoyé au vérificateur de formule.

Comme vu précédemment, cette partie a impliqué le choix du langage de par sa complexité en terme de programmation. Peu de langages permettent de faire la liaison avec Ocaml. Les formules récupérées préalablement en C# seront donc transmises via un "pont" formé par l'outil CSML. Cette étape nécessite de renvoyer également le jardin dans son nouvel état, des modifications pouvant être apportées entre deux formules.

Le vérificateur de formule en Ocaml pourra ensuite vérifier l'exactitude de la formule dans le contexte donné et renvoyer le résultat via le pont CSML à l'application en C# qui remontera alors graphiquement à l'étudiant la décision de l'application.

4 Organisation

4.1 Outils

Il a été décidé d'utiliser un gestionnaire de version pour faciliter l'accès aux documents et le travail en parallèle via des branches de développement. Le choix s'est porté sur github qui offre des dépôts privés aux étudiants.

4.2 Partage des tâches

A partir de la date de rendu officielle du cahier des charges de l'application, il reste 2 mois et demi de programmation à séquencer. Le programme s'axe sur trois grandes parties, deux complexes :

— analyse syntaxique,
— liaison Ocaml et C# avec CSLM,

et une partie plus simple mais longue :

— sauvegarde des jardins,
— programmation de l'interface graphique.

Afin de paralléliser au maximum les taches, une personne devra s'occuper de l'analyse syntaxique pendant que l'autre s'occupera de la liaison CSML. Cette partie se déroulera sur un mois et demi de programmation afin de laisser le temps de faire le plus de tests possibles. Il est important que cette partie de l'application soit la plus robuste possible.
Le dernier mois sera consacré à toute l'interface graphique et les différents modules de celle-ci seront partagés entre les programmeurs :

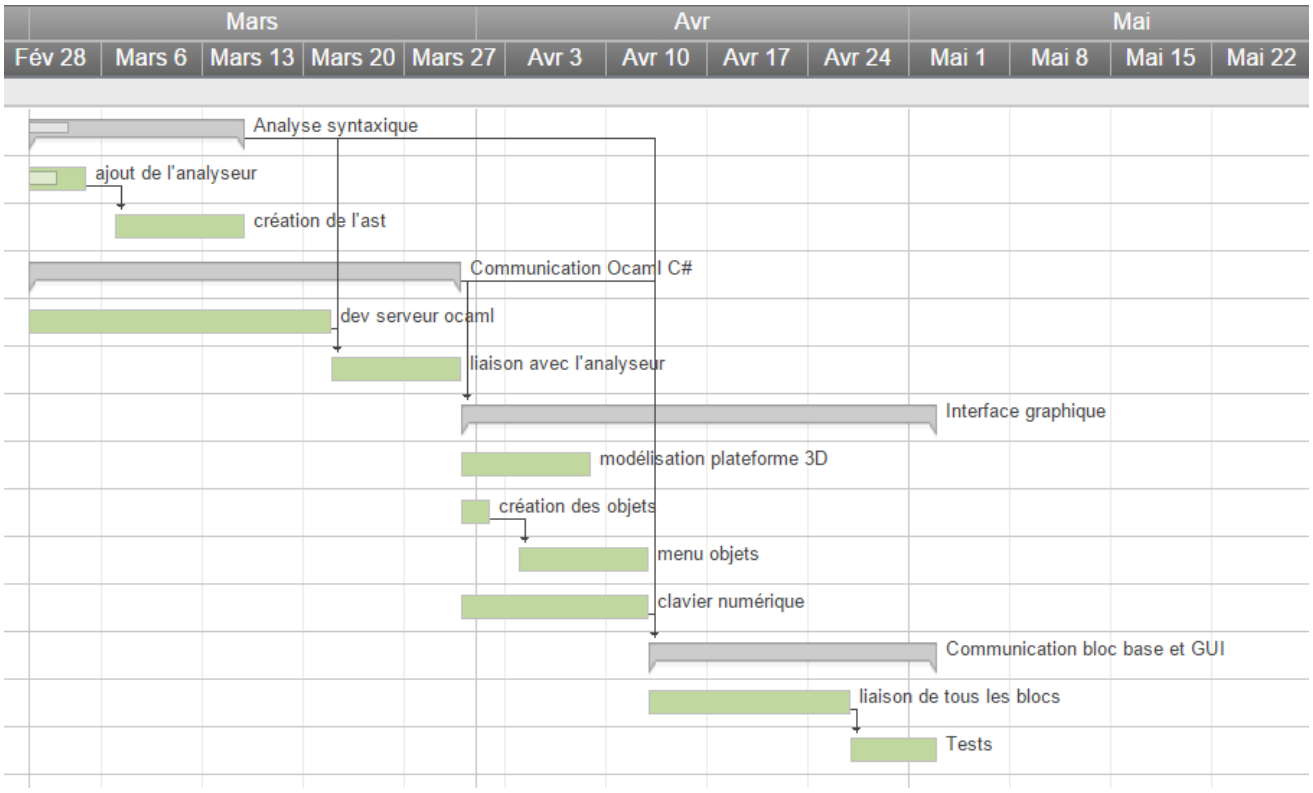
— affichage 3D/2D du jardin,
— création du clavier optique,
— modélisation des fleurs,
— création des divers menus.

Dans le cas où le programme serait entièrement fonctionnel avant la période impartie, il pourra être envisagé d'ajouter des fonctionnalités optionnelles.

Logique du premier ordre

	Nom de la tâche	Date de début	Date de fin	Durée	Prédécesseurs
1	**Analyse syntaxique**	**01/03/16**	**15/03/16**	**11j**	
2	ajout de l'analyseur	01/03/16	04/03/16	4j	
3	création de l'ast	07/03/16	15/03/16	7j	2
4	**Communication Ocaml C#**	**01/03/16**	**30/03/16**	**22j**	
5	dev serveur ocaml	01/03/16	21/03/16	15j	
6	liaison avec l'analyseur	22/03/16	30/03/16	7j	5; 1
7	**Interface graphique**	**31/03/16**	**02/05/16**	**23j**	**4**
8	modélisation plateforme 3D	31/03/16	08/04/16	7j	
9	création des objets	31/03/16	01/04/16	2j	
10	menu objets	04/04/16	12/04/16	7j	9
11	clavier numérique	31/03/16	12/04/16	9j	
12	**Communication bloc base et GUI**	**13/04/16**	**02/05/16**	**14j**	**1; 4; 11**
13	liaison de tous les blocs	13/04/16	26/04/16	10j	
14	Tests	27/04/16	02/05/16	4j	13
15					
16					

;



;

5 Fonctionnalités optionnelles

5.1 Aspects graphiques

— Animation lors de l'ajout d'une fleur et/ou temps d'inactivité de l'utilisateur trop long.
— Possibilité de saisie de formule au clavier uniquement avec un système de raccourci.

5.2 Aspects techniques

— Amélioration de l'algorithme de vérification syntaxique des formules pour indiquer où se trouve l'erreur.
— Implémentation d'un système de « retour en arrière ».