Année universitaire 2016-2017
Université de Caen Basse-Normandie

Rapport sur le deuxième semestre de TPA

Alexis Carreau
Thomas Lécluse
Emma Mauger
Théo Sarrazin
L2 Informatique

Réalisation d'un IDE en Python

Table des matières

1 Introduction **2**
1.1 Grammaire pour Lex et Yacc . . . 2
1.2 Pour ce semestre . . . 3

2 Les thèmes et les styles **4**
2.1 Changer de thème . . . 5
2.2 Gestion des thèmes . . . 6
2.3 Création de thèmes . . . 7
2.4 Style externe . . . 8

3 Informations sur le code **9**
3.1 Une autre barre de status . . . 9
3.2 Numérotation des lignes . . . 10

4 Recherche et édition du texte **10**
4.1 Selection de la ligne courante . . . 10
4.2 Selection du mot courant . . . 10
4.3 Duplication . . . 11
4.4 Recherche . . . 12
4.5 Indentation du fichier . . . 13
4.6 Commenter la sélection . . . 14
4.7 Les snippets et l'autocomplétion . . . 15

5 Traitement des projets **15**
5.1 Import d'un projet . . . 15
5.2 Informations d'un projet . . . 15

6 Fichier de configuartion XML **15**

7 Compilateur **15**

8 Insecteur d'éléments **15**

9 Personnalisation des raccourcis **16**

10 Cache des documents **16**

11 Nouvelles grammaires **16**
11.1 Grammaire Arithmétique . . . 16
11.2 Grammaire Python . . . 16

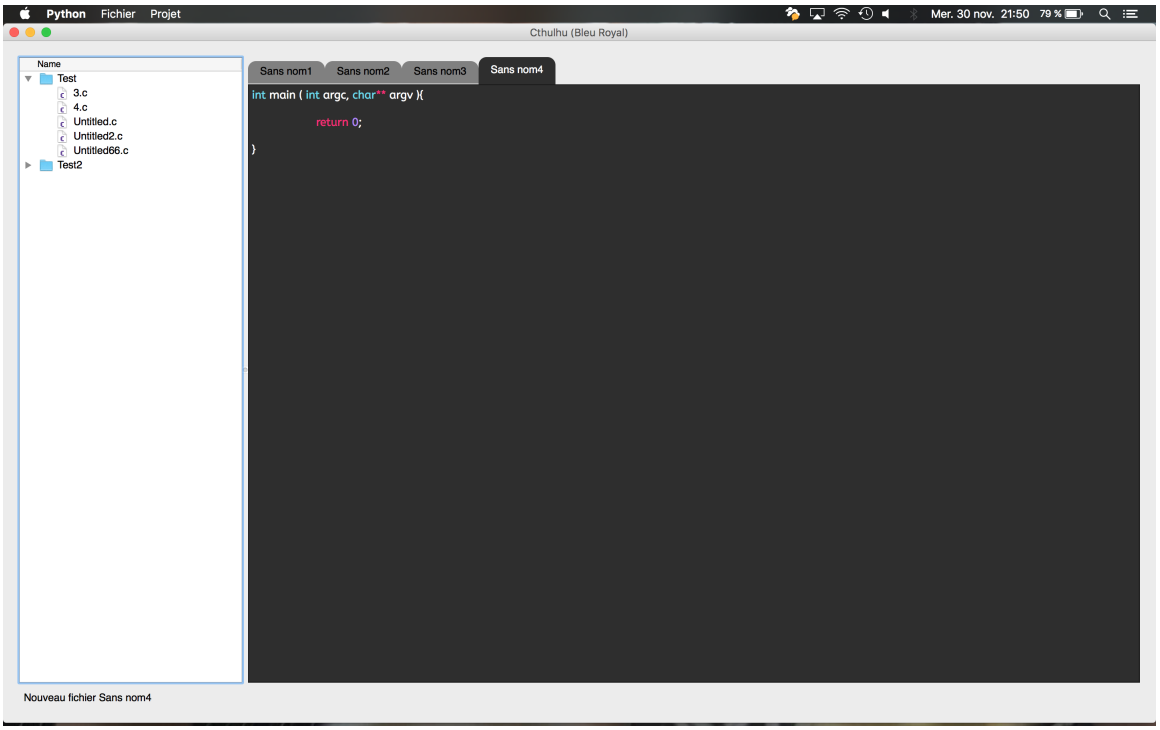
12 Langue du système en Anglais **16**

13 Ajouts bonus **16**

1 Introduction

Voici un résumé de ce que nous avions à la fin du premier semestre.

Notre IDE était capable d'ouvrir des documents avec l'extention .c ou .h à partir de projets qu'on avait créé. On pouvait ouvrir plusieurs documents, et avoir une liste d'onglets. Nous avions un navigateur de fichiers qui nous permettait de naviguer entre nos différents projets et leurs documents. Une barre de menu nous permettait d'accéder à nos différentes fonctionnalités, et on affichait à l'aide d'une barre de status différents messages répondant aux requête de l'utilisateur.



Nous colorions le contenu des documents à l'aide du logiciel Lex les différents tokens (éléments du code) selon leur fonction. Et nous analysions les documents pour détecter des erreurs de syntaxe à l'aide du logiciel Yacc. Nous utilisons pour cela une grammaire.

1.1 Grammaire pour Lex et Yacc

À l'aide du module PLY (Python Lex and Yacc), nous définissons une liste de tokens afin de déterminer tous les mots clefs du langage.

Nous avons aussi des fonctions à définir, pour Yacc afin de définir la syntaxe à suivre pour le langage (ici le langage C). Pour donner un exemple, sur les expressions de multiplication, nous indiquons qu'une multiplication peut prendre plusieurs formes.

Soit une "cast_expression", soit une expression multiplicative multipliée par une "cast_expression", soit une expression multiplicative divisée par une "cast_expression" ou soit expression multiplicative modulo une "cast_expression".

Une "cast_expression" étant définie par une expression unitaire ou bien un "type_name" entre parenthèses suivi d'une "cast_expression". Où un "type_name" est une autre fonction définissant ce que c'est.

On décrit ci-dessus toutes les formes possibles d'expressions unitaires. C'est donc ainsi de suite et en remonttant toutes les clauses que l'on définit les règles syntaxiques du langage.

```
types = [
    "char",
    "bool",
    "double",
    "enum",
    "float",
    "int",
    "long",
    "short",
    "signed",
    "unsigned",
    "void"
]
```

FIGURE 1 – Exemple de liste de tokens pour les déclarations de type en C.

```
def p_multiplicative_expression(p):
    '''multiplicative_expression : cast_expression
                                 | multiplicative_expression TIMES cast_expression
                                 | multiplicative_expression DIVIDE cast_expression
                                 | multiplicative_expression MOD cast_expression'''
```

FIGURE 2 – Différentes formes d'expressions de multiplication.

```
def p_cast_expression(p):
    '''cast_expression : unary_expression
                       | L_BRACKET type_name R_BRACKET cast_expression'''
```

FIGURE 3 – Une "cast_expression"

```
def p_unary_expression(p):
    '''unary_expression : postfix_expression
                        | INC_OP unary_expression
                        | DEC_OP unary_expression
                        | unary_operator cast_expression
                        | SIZEOF unary_expression
                        | SIZEOF L_BRACKET type_name R_BRACKET'''
```

FIGURE 4 – Une expression unitaire

1.2 Pour ce semestre

Nous avons beaucoup travaillé à étoffer et rajouter de nombreuses fonctionnalités à notre éditeur afin qu'il soit complet et qu'il ressemble à des éditeurs déjà existant au niveau du contenu.

Voici donc la liste de ce qui a été ajouté depuis les vacances de Noël :
— Différents thèmes et styles pour l'éditeur et ses éléments
— Traitement des projets
— Fichiers de configuration au format XML
— Intégration d'un compilateur
— Fonctionnalités diverses de recherche et d'édition de texte
— Inspecteur d'éléments
— Une autre barre de status pour d'autres types d'informations

— Les numéros de lignes
— Fenêtre de paramétrage des raccourcis
— Ajout de cache afin d'optimiser l'ouverture de fichiers
— Grammaire arithmétique puis grammaire python
— Ajout de l'Anglais
— Quelques bonus

À noter que nous avions réalisé des rapports au fur et à mesure de l'avancement. Nous les reprennons ici donc en les ré-adaptant à l'aspect global évidemment. Mais il peut rester des images notamment sur la barre de menu où les fonctionnalités ne sont plus placées au même endroit. Par exemple, le menu apparence est maintenant placé dans Fichier/Paramètres.

2 Les thèmes et les styles

Afin de pouvoir rendre plus personnalisable l'application, nous avons choisi de permettre la personnalisation du thème global. Voici quelques exemples de thèmes :

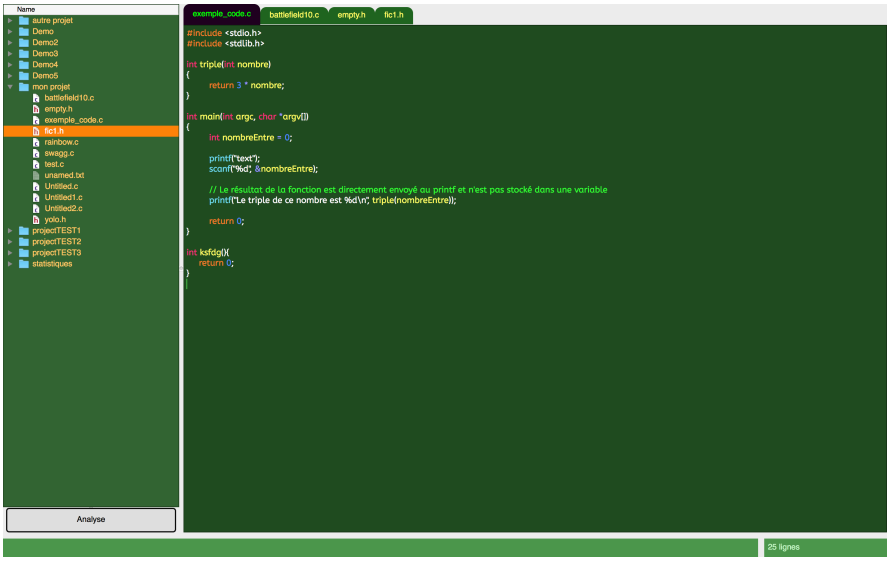


FIGURE 5 – Thème Forêt

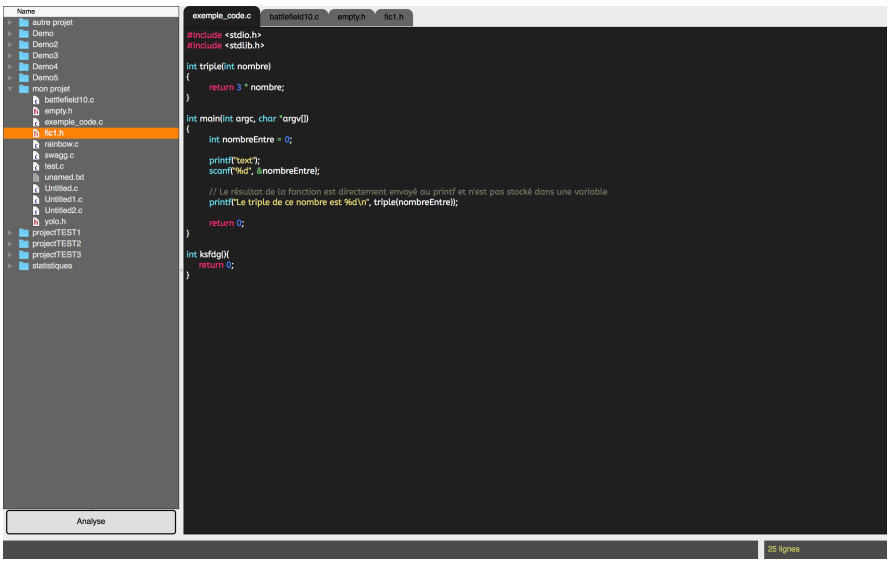


FIGURE 6 – Thème de base

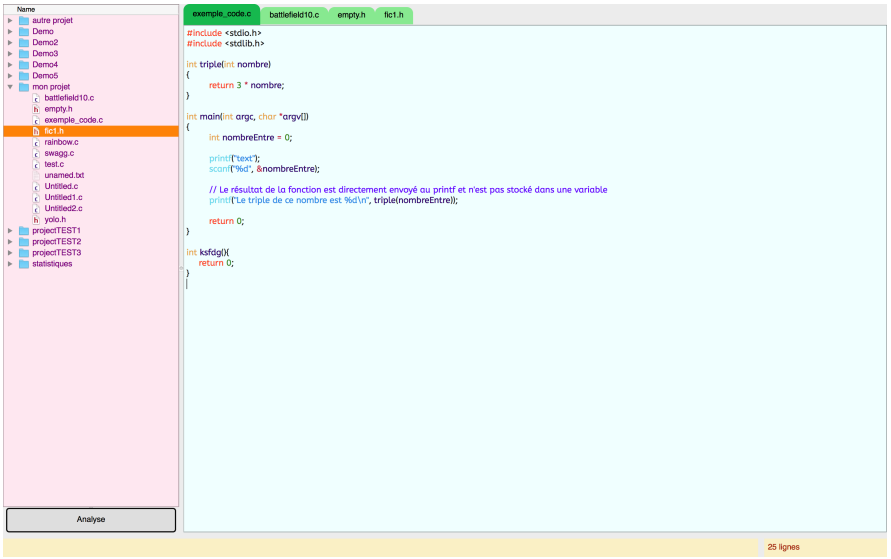


FIGURE 7 – Thème Pastel

2.1 Changer de thème

Le changement de thème est très simple. Il suffit de se rendre dans le menu "Fichier/Paramètres" puis de choisir son thème parmi ceux proposés dans les catégories clairs et sombres.

Les thèmes sont triés en fonction de si ils sont plutôt clairs ou plutôt sombres.
Lorsque l'on sélectionne son thème il est immédiatement changé, il n'y a pas besoin de relancer l'application. De plus, une petite icone apparaît à côté du thème que vous avez choisi dans la barre de menu

Le thème sélectionné est alors écrit de le fichier conf.xml qui est situé à la racine du projet. Ainsi, lorsque vous relancerez l'IDE, le dernier thème que vous avez utilisé sera rechargé.

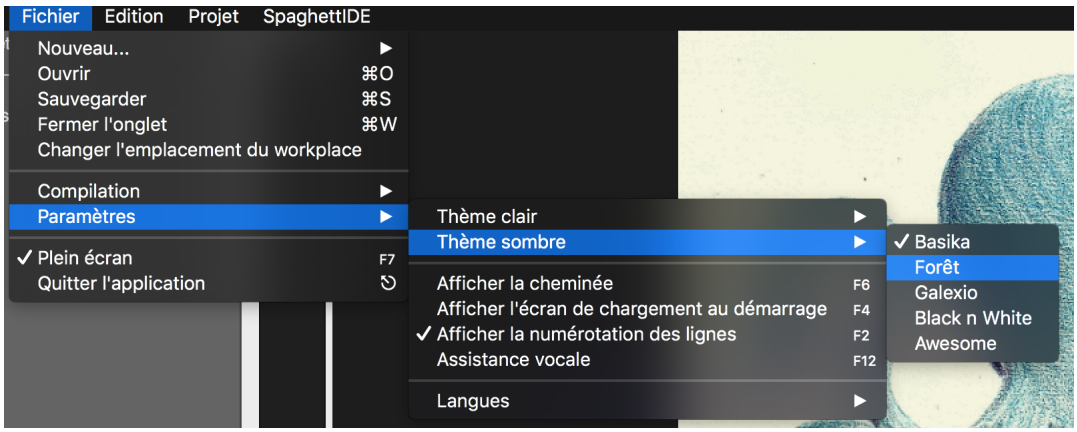


FIGURE 8 – Choix de son thème

2.2 Gestion des thèmes

Les thèmes sont regroupés dans des répertoires distincts, le tout dans le répertoire "theme" situé à la racine du projet.

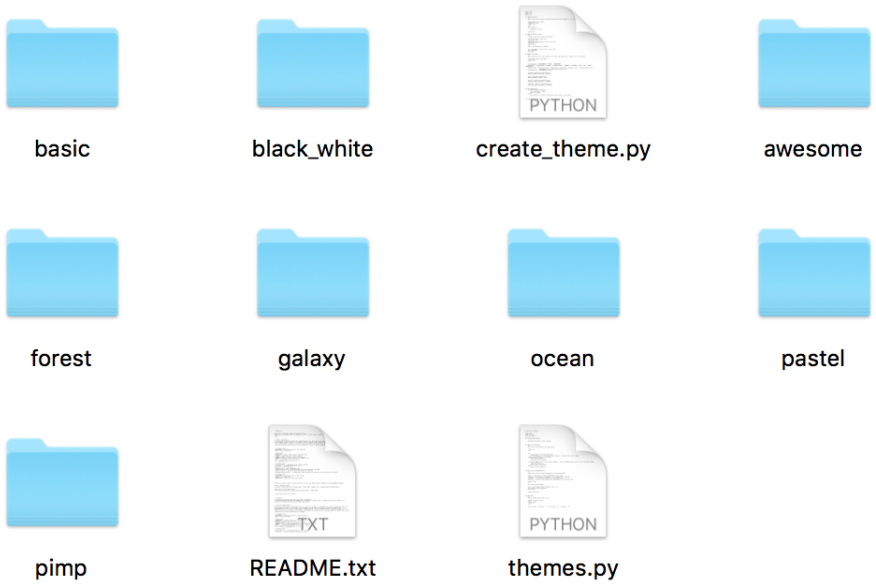


FIGURE 9 – Contenu du répertoire "theme"

Chaque répertoire de thème regroupe les fichiers en format .json. Les différents fichiers .json contiennent les couleurs en RGB d'un élément de l'interface graphique.

Le module theme.py nous permet de récupérer le thème sauvegardé (dans le fichier conf.xml) lors du chargement de l'application notament. Ici sont également contenues les méthodes permettant à l'interface d'aller chercher les couleurs qu'elle doit appliquer aux différents éléments.

Au niveau technique, nous utilisons la méthode .setStyleSheet() de QT qui peut s'appliquer à la majorité des widgets (tous dans notre cas) et qui nous permet donc de spécifier et de modifier les couleusr de fond ainsi que de police des widgets en fonction du thème choisi.

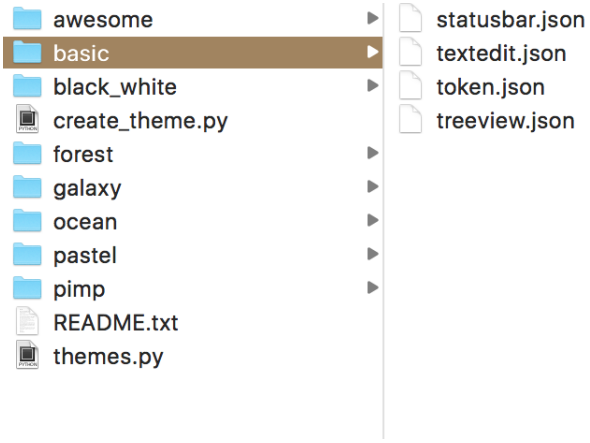


FIGURE 10 – Contenu du répertoire "basic" (le contenu est semblable pour tous les thèmes)

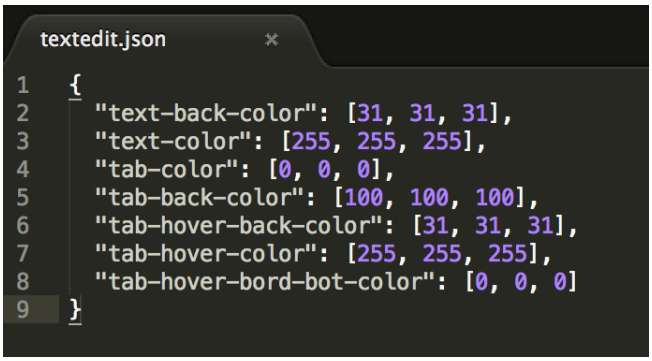


FIGURE 11 – Exemple de fichier .json pour les thèmes.

2.3 Création de thèmes

Vous pouvez utiliser les thèmes pré-définis, qui ont été pour la plupart validés et certifiés par la totalité du groupe comme étant jolis, mais vous pouvez aussi créer vos propres thèmes.

Le script "createtheme.py" vous permet cela, et la démarche à suivre est expliquée dans le fichier README.txt.

Pour résumer, on lance le fichier createtheme.py via un terminal en spécifiant le nom du thème. Tapez par exemple : "python3 createtheme.py monNouveauTheme" et cela créera un répertoire "monNouveauTheme" qui contiendra les fichiers nécessaires à la gestion de votre thème. Ouvrez ensuite les fichiers .json et définissez vos propres couleurs (par défaut tout est noir).

Une fois cela fait, vous devrez ajouter deux lignes dans le module menu.py pour que votre thème apparaisse dans la séléction.

```
nomTheme = MyAction(parent, "&monNouveauTheme", "monNouveauTheme", lambda: self.__change_theme_to("monNouveauTheme"))
```

FIGURE 12 – Où nomTheme est le nom de la variable pour le thème, et monNouveauTheme le nom que vous avez donné à votre thème

```
self.set_group(nomTheme, groupe_theme, apparence_menu, "monNouveauTheme")
```

FIGURE 13 – Où nomTheme est le nom de la variable pour le thème, et groupetheme le groupe (clair ou sombre) auquel appartient votre thème

Votre thème apparaît maintenant dans la barre de menu et il est sélectionnable !

2.4 Style externe

Dans le répertoire "gui" qui contient tout ce qui est relatif à l'interface graphique, il y a un répertoire "style" qui regroupe des éléments auxquels on applique également des feuilles de styles (via la méthode setStyleSheet() des widgets dans QT ; de même que pour les thèmes).

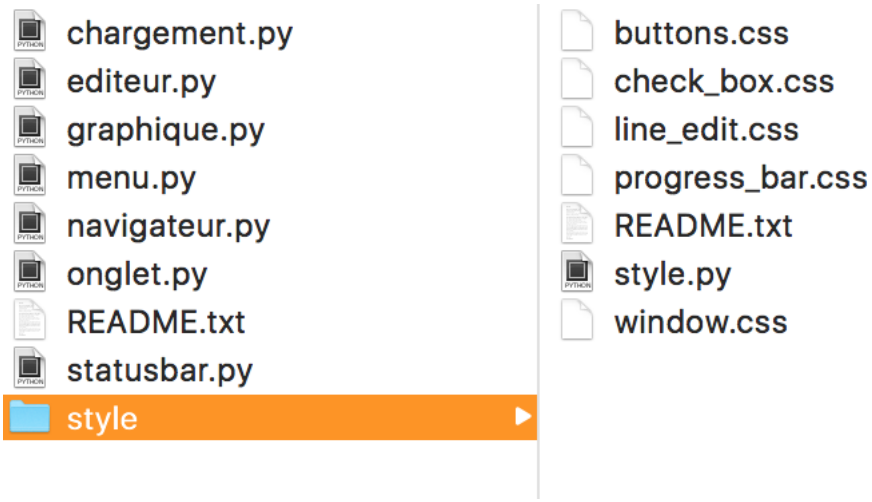


FIGURE 14 – Contenu du répertoire style

Chaque document .css contient le style relatif à des éléments. Nous retrouvons ici le style appliqué :

— aux boites de dialogue

FIGURE 15 – Exemple de la fermeture de l'IDE : une popup qui apparaît demandant la confirmation.

— aux boutons, à qui on inverse les couleurs lorsqu'on les survole

FIGURE 16 – Style appliqué à un bouton normal.

FIGURE 17 – Le même bouton lorsqu'il est survolé par la souris.

— à la barre de progression

FIGURE 18 – La barre de status (voir ci-dessous).

3 Informations sur le code

3.1 Une autre barre de status

Nous avons également travaillé à étoffer l'interface. Nous avons rajouté une seconde barre de status (en bas à droite), servant à afficher des informations sur le code lui-même :

FIGURE 19 – Le nombre de lignes du fichier courant

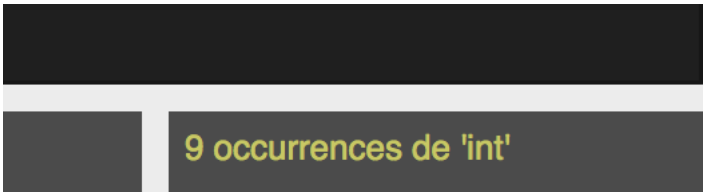


FIGURE 20 – Le nombre d'occurences d'une recherche effectuée

Nous utilisons de plus cet emplacement (en bas à droite) pour afficher une barre de status qui sert pour le moment uniquement à indiquer la progression lors du chargement de projet (Yacc lisant tous les fichiers afin de récupérer les différentes fonctions à travers les modules, cela peut prendre plusieurs secondes).

L'ouverture de projets est maintenant beaucoup plus rapide qu'au début du semestre, notamment grâce à l'utilisation du cache. La barre de progression apparaît toujours, mais c'est souvent très rapide.

FIGURE 21 – Barre de progression lors du chargement d'un projet de l'utilisateur

3.2 Numérotation des lignes

4 Recherche et édition du texte

4.1 Selection de la ligne courante

Dans un premier temps, nous avons ajoutés une fonction permettant de selectionner la ligne où ce trouve le curseur. Pour cela, nous recuperons l'objet QTextCursor de notre class Editeur (héritant de QTextEdit) puis nous utilisons la méthode **select** de cet objet qui nous permet de selectionner du texte dans notre Editeur, cette méthode prend en paramètre une méthode de selection, QTextCursor.LineUnderCursor dans notre cas. La ligne ou ce trouve notre cuseur va donc être selectionner. Pour appliquer ces modifications, nous devons appliquer notre objet QTextCursor à notre Editeur, pour cela on utilise la méthode **setTextCursor** de l'objet Editeur et on lui passe en paramètre notre QTextCursor.

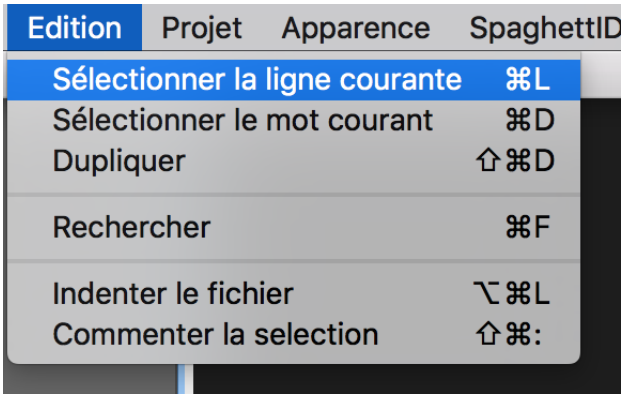


FIGURE 22 – Action du menu permettant la selection de la ligne courante

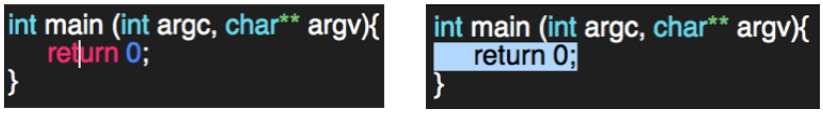


FIGURE 23 – Résultat de l'utilisation de la fonction selection de la ligne courante

4.2 Selection du mot courant

Pour l'ajout de la selection du mot courant, la démarche est exactement la même que pour la selection de la ligne courante, nous devons simplement changer la méthode de selection, passant de QTextCursor.LineUnderCursor à QTextCursor.WordUnderCursor, afin de ne plus sélectionner la ligne mais le mot présent au niveau du curseur.

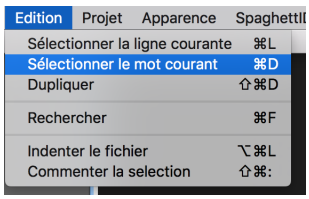


FIGURE 24 – Action du menu permettant la selection du mot courant

FIGURE 25 – Résultat de l'utilisation de la fonction selection du mot courant

4.3 Duplication

Pour l'ajout de la duplication du texte, nous avons choisi de différencier deux cas, le premier où rien n'est selectionné et le second où du texte est déjà selectionné. Dans le premier cas toutes la ligne est dupliquée et dans le second seulement la partie selectionnée est dupliquée.

Pour cela, nous recupperons une nouvelle fois le QTextCursor de notre Editeur, puis pour savoir dans quel cas nous sommes on utilise la méthode selectedText de l'objet QTextCursor. Ainsi si aucun text n'est selectionné nous selectionnons la ligne courante de le même façon que précédement de plus on assigne la valeur
n à la variable **return_** en effet si on duplique une ligne entière, on ajoute on retoure à la ligne entre la selection d'origine et la partie dupliquée. Puis on ajoute le texte dans notre objet Editeur grâce à la méthode **inserText** avec en paramètre la selection du QTextCursor (recuperrée grâce à la méthode **selectedText**) suivi de la variable **return_** elle même suivi de la selection du QTextCursor.

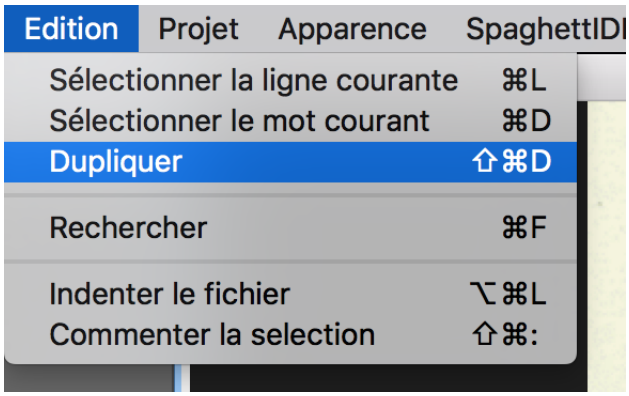


FIGURE 26 – Action du menu permettant de dupliquer

```
int main (int argc, char** argv){
    return 0;
    return 0;
}
```

```
int main (int argc, char** argv){
    return 0;
}
```

FIGURE 27 – Résultat de l'utilisation de la fonction permettant de dupliquer

4.4 Recherche

Pour la recherche dans le document, nous avons decidés d'ajouter une boite de dialogue permettant d'entrée le texte à rechercher. Pour cela nous avons créés une classe SearchDialog (héritant de QDialog), lors de l'affichage de cette boite de dialogue nous utilisons la méthode **exec**, qui rend impossible l'interaction avec la fenetre en arrière plan tant que la boite de dialogue est ouverte.

Cette boite de dialogue nous permer de taper le texte à rechercher, de choisir si on recherche en avant ou en arrière , mais aussi si on veut être sensible à la case.

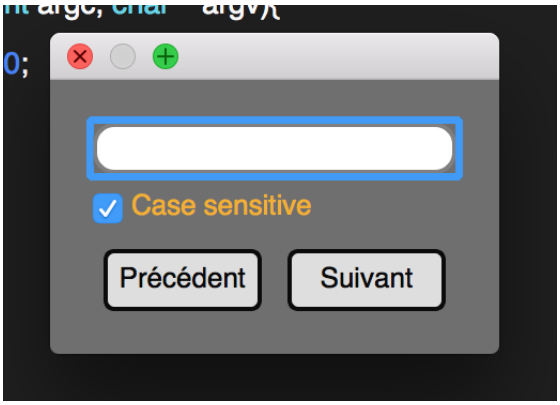


FIGURE 28 – Boite de dialogue relative à la recherche

Pour la recherche on utilise la méthode **find** de notre objet Editeur. Cette méthode prend en paramètre le texte à rechercher, suivit de differents drapeaux. Dans notre cas nous utilisons le drapeau permettant d'executer la rechercher en arrière et le drapeau permettant de faire la recherche en étant sensible à la case (respectivement les drapeaux **QTextDocument.FindBackward** et **QTextDocument.FindCaseSensitively**)

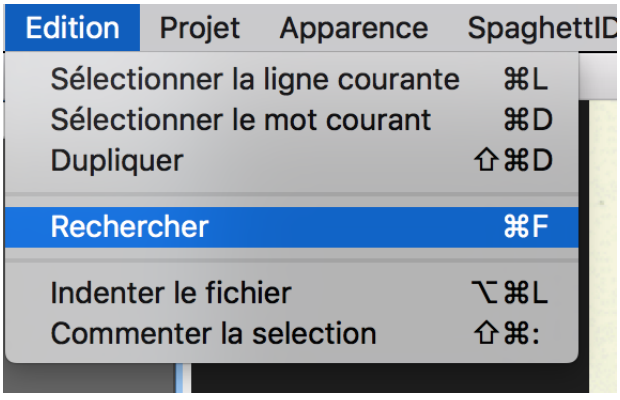


FIGURE 29 – Action du menu permettant de rechercher

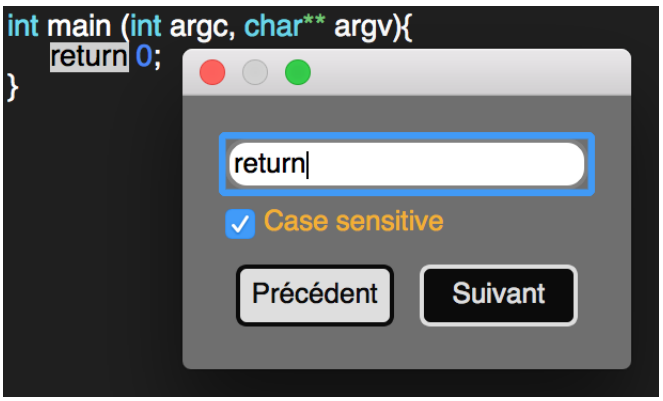


FIGURE 30 – Résultat de l'utilisation de la fonction permettant de rechercher

4.5 Indentation du fichier

Pour l'indentation du document, nous allons changer son contenue (ajout/retrait de tabulation) nous devons donc stocker la position courante du curseur (grâce à la méthode **blockNumber** de l'objet QTextCursor). Par la suite on recupère le contenue du document grâce à la méthode **toPlainText** de l'objet Editeur. On créer une variable **indent_level**, qui contient le niveau courant d'indentation, puis on parcoure toutes les lignes de notre document, si le ligne contient le charactère "}", on retire 1 au niveau d'indentation puis on change la ligne pour ajouter au debut de cette dernière **indent_level** fois une tabulation, puis on ajoute 1 au niveau d'indentation si la ligne contient "{". Pour finir on defini le nouveau texte ainsi obtenu comme texte de notre document avec la méthode **setPlainText** et on replace le curseur au bon endroit.

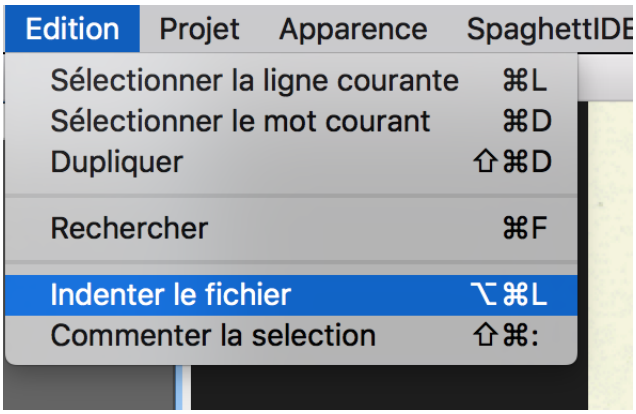


FIGURE 31 – Action du menu permettant d'indenter le fichier

FIGURE 32 – Résultat de l'utilisation de la fonction permettant d'indenter le fichier

4.6 Commenter la sélection

De la même façon que pour la duplication du texte, nous avons séparés cette action en deux cas, soit du texte est sélectionné soit rien n'est selectionné. Dans le cas ou du text est selectionné nous commenterons seulement à partir du debut de la selection. Si plusieurs lignes sont selectionnées elles seron evidement toutes commentées. Si il n'y a pas de texte selectionné, on commente la ligne courante.

Dans un premier temps, comme pour la duplication, si rien n'est selectionné on selectionne la ligne courante, puis on sauvegarde le text selectionné que l'on recupère grâce à la méthode **selectedText**, puis on supprime le texte selectionné avec la méthode **removeSelectedText**. Pour savoir si le texte est deja commenté, nous parcourons toutes les lignes, si une des lignes ne commence pas par "//" la selection est considérée comme non commenté. Par la suite nous parcourons chaque ligne du texte puis pour chaque ligne nous ajoutons/retirons les charactères "//" en fonction de si le texte et oui ou non déjà commenté.

Puis on ajoute le texte ainsi modifié à notre document en utilisant la méthode **insertText** (de l'objet QTextCursor).

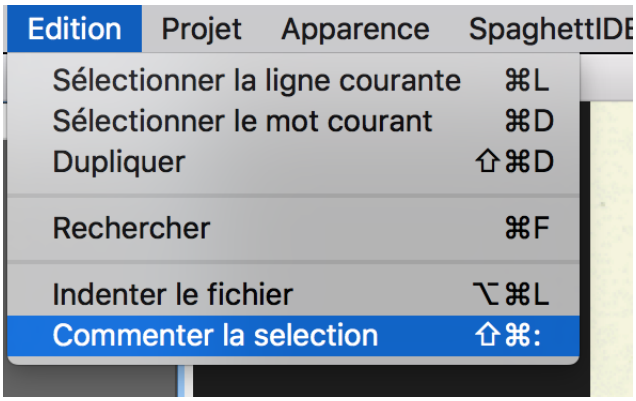


FIGURE 33 – Action du menu permettant de commenter le fichier

FIGURE 34 – Résultat de l'utilisation de la fonction permettant de commenter le fichier

4.7 Les snippets et l'autocomplétion

5 Traitement des projets

5.1 Import d'un projet

5.2 Informations d'un projet

6 Fichier de configuartion XML

7 Compilateur

8 Insecteur d'éléments

Lorsqu'un document est ouvert, nous avons la possibilité d'afficher l'inspecteur d'éléments. Pour cela, nous avons le bouton "Navigateur" sous le navigateur de fichiers, et lorsque l'on clique dessus, on affiche l'inspecteur à la place. De même si on reclique dessus, on ré-affiche le navigateur.

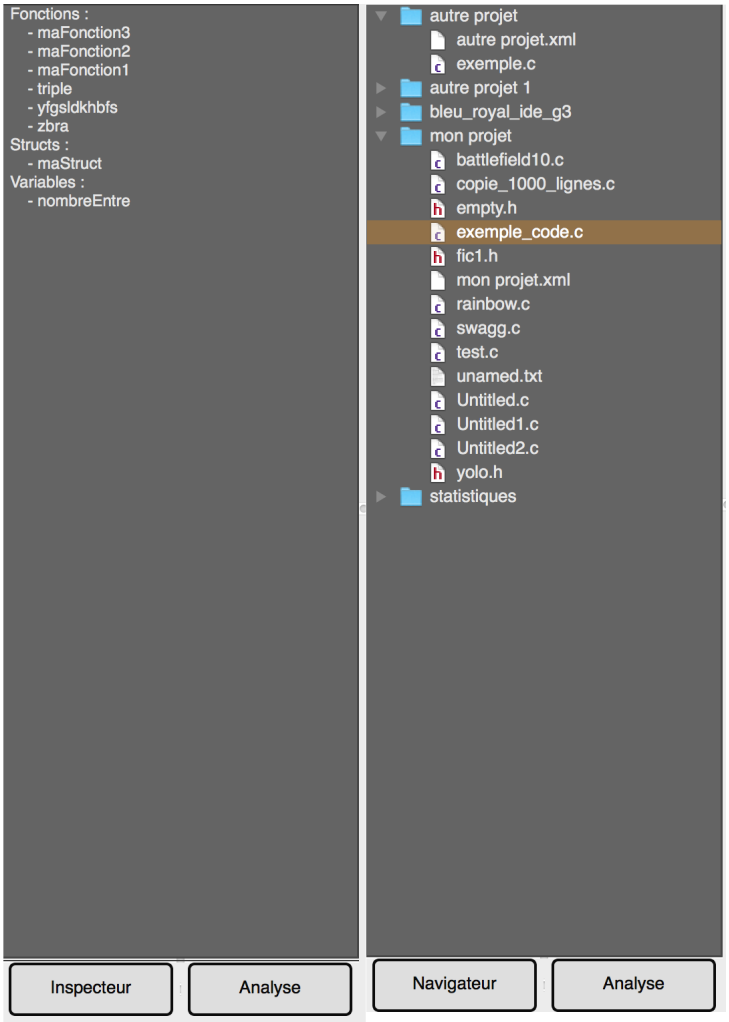


FIGURE 35 – Respectivement l'inspecteur et le navigateur de fichiers sont affichés

Au niveau du code, le navigateur de fichiers est créé à l'aide d'un QTreeView (comme au

premier semestre), et il est remplacé par un QTextEdit qui constitue l'inspecteur lorsqu'on clique sur le bouton.

On affiche dans l'inspecteur toute la structure du code grâce au logiciel Yacc, qui peut nous renvoyer des listes contenant pour un fichier :
— Ses variables
— Ses fonctions
— Ses struct (en C) ou ses Class (en Python)

Lorsque l'on clique sur un élément par exemple un nom de variable, cette variable est sélectionné dans le document courant.

9 Personnalisation des raccourcis

10 Cache des documents

11 Nouvelles grammaires

11.1 Grammaire Arithmétique

11.2 Grammaire Python

12 Langue du système en Anglais

13 Ajouts bonus