

Antoine de ROQUEMAUREL (antoine.de-roquemaurel@univ-tlse3.fr)
Fabrice VALLEIX (valleix.fabrice@gmail.com)
Groupe 2.2

Dossier final

Projet logiciel

Avant-propos

Ce dossier concerne un projet logiciel développé en langage C : Un jeu de Boggle.

Il à été conçu par Antoine de ROQUEMAUREL et Fabrice VALLEIX dans le cadre du module *projet logiciel* de la L2 Informatique de l'université Toulouse III – Paul Sabatier.

Ce dossier concerne la conclusion du projet, la manière de compiler et exécuter le programme, nos méthodes de gestions de projet et les différents outils que nous avons utilisés afin de développer ce logiciel.

Le dossier de conception mis à jour par rapport à ce que nous avons effectué est disponible en annexe page 19.

Table des matières

1	Compilation et exécution	5
1.1	Compilation	5
1.2	Exécution	5
2	Qualité du code	7
2.1	Convention de codage	7
2.2	Documentation	7
2.3	Utilisation de <i>Sonar</i>	7
3	Tests	9
3.1	Tests unitaires avec <i>CUnit</i>	9
4	Gestion de projet	12
4.1	Un outil de gestion de projet : <i>Redmine</i>	12
4.2	Un logiciel de versionnement : Git	13
A	Convention d'écriture en C	14
A.1	Le nommage	14
A.2	L'indentation	15
A.3	Les accolades	16
B	Diagramme de Gantt	17
C	Dossier de conception (Mis à jour)	19

Compilation et exécution

1.1 Compilation

La compilation du projet se fait à l'aide de l'utilitaire **Make**, ainsi la simple commande **make** à la racine du projet suffit à compiler le projet.

Cependant, afin de pouvoir compiler le projet, il est indispensable de posséder la bibliothèque *Ncurses* sur sa machine, sinon la compilation ne fonctionnera pas.

Il est possible de l'installer avec la commande **apt-get install libncurses5-dev** sur les Linux utilisant le gestionnaire de paquet de Debian.

Les tests quant à eux se compilent à l'aide de la commande **make test**, cette commande va compiler puis exécuter tous les tests, cependant si vous ne possédez pas *CUnit* sur votre machine, il est également indispensable de taper la commande suivante afin de signaler au système l'emplacement de la bibliothèque.

```
| LD_LIBRARY_PATH=$LD_LIBRARY_PATH: 'pwd' /Cunit/lib && export LD_LIBRARY_PATH
```

Listing 1.1 – Commande execution tests

Cette commande doit être employée à chaque ouverture d'une nouvelle console, la variable étant attachée à une seule console.

1.2 Exécution

Afin d'exécuter notre application, vous devez utiliser l'exécutable **./boggle**, son utilisation est la suivante :

```
| ./boggle --solveur|--texte|--ncurses [--grilleFixe]
```

Afin d'appeler les différentes fonctionnalités du programme, il est nécessaire de faire passer un paramètre, celui-ci peut prendre la forme d'une des trois chaînes de caractères ci-dessous. Le dernier paramètre est facultatif.

--solveur Correspond à la version 1 du projet. Afin d'appeler la version 1 de l'application, l'exécutable doit être appelé à l'aide de l'argument **--solveur**

Dans cette version, une grille carrée de la taille demandée par l'utilisateur est générée, en tenant compte de la fréquence des lettres dans la langue Française. Une fois la grille générée, la position d'une case est demandée à l'utilisateur, l'utilisateur entre donc les deux coordonnées, et tous les mots commençant par cette case seront affichés à l'écran.

Attention, les coordonnées de la grille commencent à zéro.

--texte Afin d'appeler la version 2, l'exécutable doit être appelé à l'aide de l'argument **-text**

Cette version fait appel à la version 1, en effet, au lancement de l'application, il est de nouveau demandé la taille de la grille, ensuite l'intégralité de la grille générée est résolue. Une fois cette étape franchie, l'utilisateur a 3 minutes pour entrer le plus de mots possibles, l'application lui signalant si le mot est accepté ou non, une fois ce temps imparti, la solution est affichée, puis le nombre de points obtenu par le joueur.

--ncurses Afin d'appeler la version 3, l'exécutable doit être appelé à l'aide de l'argument **--ncurses**

Cette version suit le même principe que la version précédente, à la différence près qu'elle utilise la bibliothèque *Ncurses*. Ainsi, la saisie des mots se fait dorénavant avec les touches fléchées du clavier, et espace pour ajouter

une lettre au mot. Pour proposer le mot surligné, la touche entrée doit être appuyée. Il est également possible de demander le nombre de mots commençant par la case sélectionnée à l'aide de la touche h.

Une fois les 3 minutes écoulées, les mots proposés par l'utilisateur et le nombre de points obtenus sont affichés, il est proposé à l'utilisateur d'afficher la solution complète.

--grilleFixe Celui-ci est optionnel et permet de signaler au programme que vous souhaitez utiliser une grille prédéfinie, ainsi l'utilisation de ce dernier paramètre lancera systématiquement le programme avec la même grille, et donc la même solution.

Qualité du code

Durant ce projet, nous avons essayés d'avoir le code le plus lisible et réutilisable possible. Ainsi, nous avons utilisés plusieurs techniques.

2.1 Convention de codage

La première chose afin d'avoir un code propre et uniforme était de nous fixé des conventions de code. En effet, nos styles de programmations étant différents, il était important de nous mettre d'accord. Ces conventions ont été écrite sur un wiki afin que nous puissions tout deux les consulter, celles-ci sont disponibles annexe A.

Ces conventions fixent la mise en forme du code, elles contiennent principalement l'écriture des noms de variables ou paramètres, les noms de fonctions, la forme de l'indentation.

2.2 Documentation

Afin d'avoir un code clair pour nous, tout en pouvant document notre code pour une personne exterieur, nous avons utilisé un outil appelé *Doxygen*. De la même manière que Javadoc, nous devons documenter les entêtes de fonctions, structures ou variable avec une syntaxe précise, ces commentaires nous permettent de relire facilement le code en le comprenant bien. De plus, l'analyse de ceux-ci avec *doxygen* permet de générer une documentation HTML ou PDF ¹).

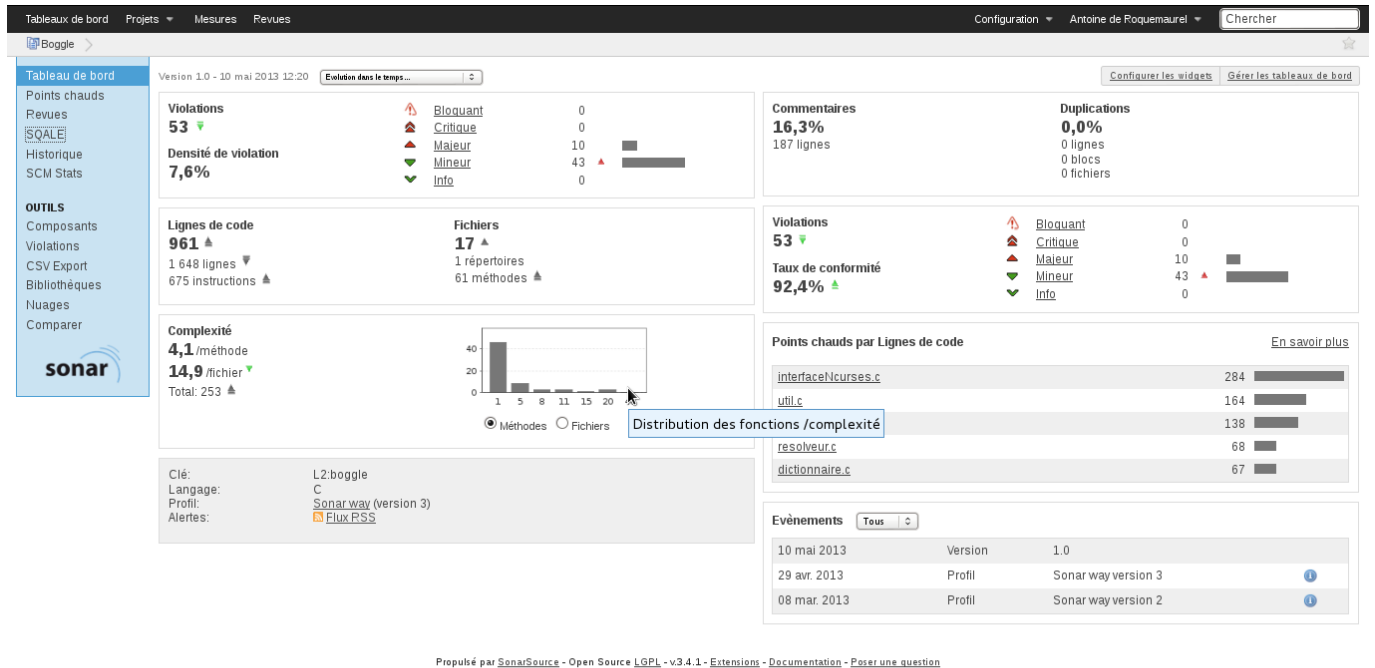
La documentation du projet est ainsi disponible :

- Sur le Web à l'adresse <http://documentation.joohoo.fr/L2/jeuDeBoggle/index.html>.
- En PDF, dans l'archive donnée avec ce projet dans `doc/documentation.pdf`.
- Tous les fichiers HTML sont disponibles dans `doc/html/`.

2.3 Utilisation de Sonar

Afin d'avoir le code le plus propre possible, nous avons utilisé un outil appelé Sonar (Cf figure 2.1), celui-ci nous signale lorsque nous ne respectons pas ses conventions, nous donne les complexités du code, analyse la duplication de code, ... Au niveau de ses conventions de programmation, il existe une multitude de règles à ne pas violer comme l'utilisation de `break` en dehors de `switch`, une complexité trop importante pour une fonction, une boucle `while` pouvant être transformée en `for` etc...

1. Celle-ci est générée à l'aide de L^AT_EX

FIGURE 2.1 – Affichage du tableau de bord de *Sonar*

Sur la figure 2.1, nous pouvons observer le nombre de ligne effective du programme (nombres de lignes sans compter les commentaires), le nombre de fichiers, la complexité par fonction/fichier, la duplication de code, les violations par rapport aux règles de Sonar.

Tests

3.1 Tests unitaires avec CUnit

Tous les tests unitaires du projet ont été effectués à l'aide de *CUnit*. *CUnit* est une bibliothèque en C permettant d'effectuer des tests unitaires simplement.

De plus, étant donné que nous utilisons l'IDE¹ *Netbeans*, celui-ci intégrait parfaitement cette bibliothèque. Ainsi, c'est lui qui nous a généré une partie du code permettant les tests, également il montrait une barre de progression comme le montre la figure 3.1.

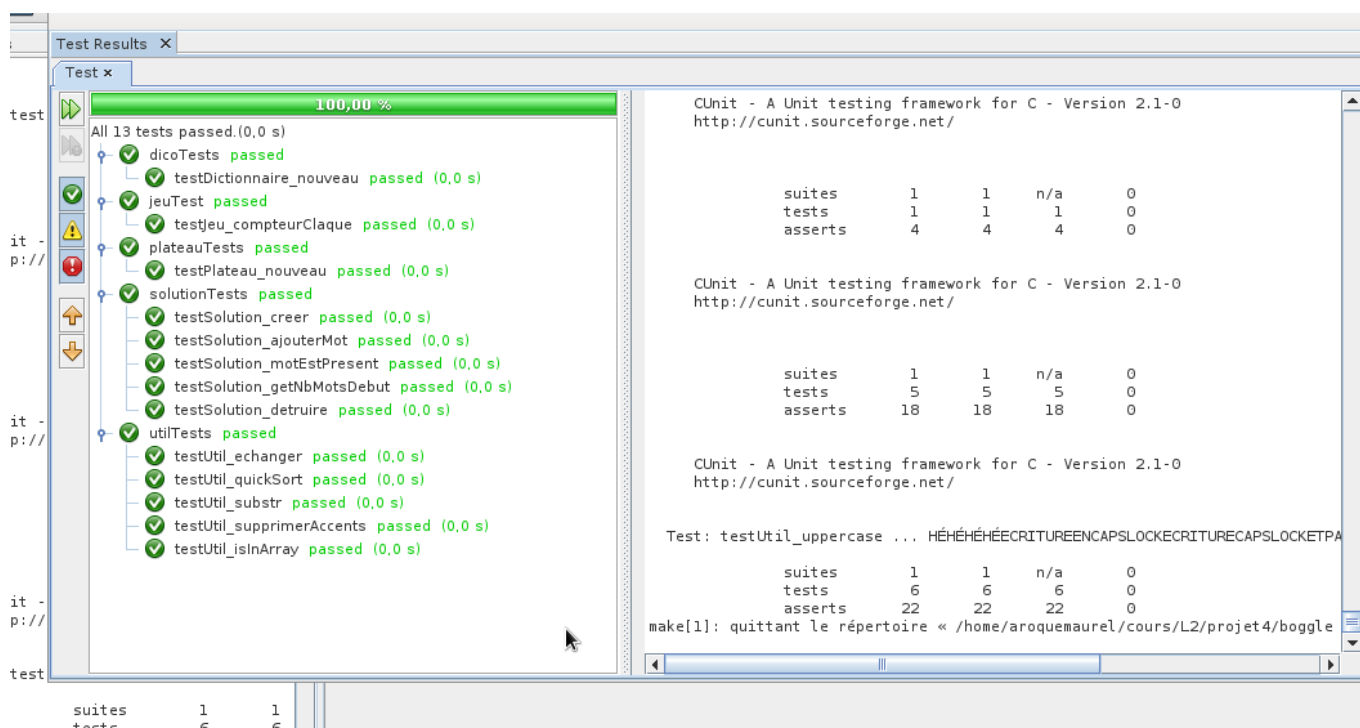


FIGURE 3.1 – Affichage des demandes dans *Redmine*

Afin de développer un tests en *CUnit*, il faut utiliser des macros spécifique, celle-ci retournant vrai si le test passe ou faux sinon. Le listing 3.1 montre les principales macros.

```

1 CU_ASSERT(bool); // Expression booléenne vraie
2 CU_ASSERT_FALSE(bool); // Expression booléenne faux
3
4 CU_ASSERT_EQUAL(actual, expected); // Nombres egaux
5 CU_ASSERT_NOT_EQUAL(atual, expected); // Nombres différents
6
7 CU_ASSERT_PTR_NOT_NULL(ptr); // Pointeur non null

```

1. Integrated Development Environment

```
8 CU_ASSERT_PTR_NULL(ptr); // Pointeur null
9
10 CU_ASSERT_STRING_EQUAL(actual, expected); // Chaine égales
11 CU_ASSERT_STRING_NOT_EQUAL(actual, expected); // Chaines différentes
```

Nous avons ainsi utiliser ces macros pour tester le projet, bien que nous avons utilisé Netbeans afin de vérifier que les différentes tests passent, il est possible de les exécuter en console. Pour cela, il faut effectuer la manipulation comme expliquée section 1.1 dans le listing 1.1.

Il est donc possible soit d'effectuer tous les tests à la suite à l'aide de la commande `make test`, qui va compiler les tests puis les exécuter, comme le montre le listing 3.1.

```
1 passed
2   Test: testJeu_compteurClaque ... passed
3   Test: testJeu_proposerMot ... 321KLJDFAIREERRADENIERpassed
4
5 --Run Summary: Type      Total      Ran   Passed   Failed
6                  suites      1        1      n/a       0
7                  tests       3        3        3       0
8                  asserts     11       11       11       0
9
10
11   CUnit - A Unit testing framework for C - Version 2.1-0
12   http://cunit.sourceforge.net/
13
14
15 Suite: plateauTests
16   Test: testPlateau_nouveau ... passed
17
18 --Run Summary: Type      Total      Ran   Passed   Failed
19                  suites      1        1      n/a       0
20                  tests       1        1        1       0
21                  asserts     4        4        4       0
22
23
24   CUnit - A Unit testing framework for C - Version 2.1-0
25   http://cunit.sourceforge.net/
26
27
28 Suite: solutionTests
29   Test: testSolution_creer ... passed
30   Test: testSolution_ajouterMot ... passed
31   Test: testSolution_motEstPresent ... passed
32   Test: testSolution_getNbMotsDebut ... passed
33   Test: testSolution_detruire ... passed
34
35 --Run Summary: Type      Total      Ran   Passed   Failed
36                  suites      1        1      n/a       0
37                  tests       5        5        5       0
38                  asserts     18       18       18       0
39
40
41   CUnit - A Unit testing framework for C - Version 2.1-0
42   http://cunit.sourceforge.net/
43
44
45 Suite: utilTests
46   Test: testUtil_echanger ... passed
47   Test: testUtil_quickSort ... passed
48   Test: testUtil_substr ... passed
49   Test: testUtil_supprimerAccents ... passed
50   Test: testUtil_isInArray ... passed
```

```
51 | Test: testUtil_uppercase ... HÉHÉHÉHÉ
52 |      ECRITUREENCAPSLOCKE CRITURECAPSLOCKETPASCAPSLOCKE CRITURETOUTENLOWERCASEpassed
53 | --Run Summary: Type      Total      Ran   Passed   Failed
54 |                  suites      1        1      n/a       0
55 |                  tests       6        6        6       0
56 |                  asserts     22       22       22       0
```

Listing 3.1 – Execution des tests

Il est également possible d'exécuter les tests modules par module. Les exécutable des tests sont présents avec leur source dans le dossier `tests/`.

Gestion de projet

Pour ce projet, nous étions deux à travailler dessus, ainsi nous avons utilisé plusieurs techniques afin de se coordonner et de limiter les problèmes. Ceci n'est pas notre premier projet ensemble, notre travail en fut simplifié.

4.1 Un outil de gestion de projet : Redmine

Pour le projet, nous avons utilisé *Redmine*, une plateforme web de gestion de projet (Cf figure 4.1). Elle nous a permis de simplifier le travail, et de ne rien oublier. En effet, nous pouvons créer des tâches, signaler qu'elles sont en cours/terminées/en tests, leur donner des dates limites, les affecter à une personne etc... Ainsi lorsque l'un de nous commençait une tâche, il le signalait sur le *redmine*, ce qui permettait de tenir au courant son binôme de ses actions et de l'avancée du projet.

	#	Tâche parente	Tracker	% réalisé	Statut	Sujet	Assigné à	Mis-à-jour
<input type="checkbox"/>	249		Document	<div></div>	Terminée	Spécifications		13-03-2013 17:31
<input type="checkbox"/>	250		Document	<div></div>	Terminée	Conception		05-04-2013 14:34
<input type="checkbox"/>	276		Module	<div></div>	En cours	Utile		29-04-2013 19:27
<input type="checkbox"/>	277		Module	<div></div>	Terminée	Plateau	Fabrice Valleix	29-04-2013 19:20
<input type="checkbox"/>	284	Module-#277: Plateau	Tâche	<div></div>	Terminée	↳ Créer plateau	Antoine de Roquemaurel	16-04-2013 16:11
<input type="checkbox"/>	285	Module-#277: Plateau	Tâche	<div></div>	Terminée	↳ Générer grille de Boggle en fonction de la taille	Antoine de Roquemaurel	16-04-2013 16:11
<input type="checkbox"/>	278		Module	<div></div>	Terminée	Dictionnaire	Fabrice Valleix	16-04-2013 16:11
<input type="checkbox"/>	286	Module-#278: Dictionnaire	Tâche	<div></div>	Terminée	↳ Créer dictionnaire	Antoine de Roquemaurel	16-04-2013 16:09
<input type="checkbox"/>	287	Module-#278: Dictionnaire	Tâche	<div></div>	Terminée	↳ mot est dans le dictionnaire	Antoine de Roquemaurel	16-04-2013 16:10
<input type="checkbox"/>	288	Module-#278: Dictionnaire	Tâche	<div></div>	Terminée	↳ Obtenir probabilité d'apparition d'une lettre	Fabrice Valleix	08-04-2013 00:10
<input type="checkbox"/>	279		Module	<div></div>	Terminée	Resolveur	Antoine de Roquemaurel	16-04-2013 16:11
<input type="checkbox"/>	299	Module-#279: Resolveur	Tâche	<div></div>	Terminée	↳ Résoudre une grille	Antoine de Roquemaurel	16-04-2013 16:10
<input type="checkbox"/>	300	Module-#279: Resolveur	Tâche	<div></div>	Terminée	↳ mot est dans la grille	Fabrice Valleix	16-04-2013 16:10
<input type="checkbox"/>	301	Module-#279: Resolveur	Tâche	<div></div>	Terminée	↳ Nombre de mots commençant par une séquence de lettre	Fabrice Valleix	16-04-2013 16:10
<input type="checkbox"/>	280		Module	<div></div>	Terminée	Jeu	Fabrice Valleix	30-04-2013 18:36
<input type="checkbox"/>	290	Module-#280: Jeu	Tâche	<div></div>	Terminée	↳ lancer timer	Fabrice Valleix	16-04-2013 16:12
<input type="checkbox"/>	297	Module-#280: Jeu	Tâche	<div></div>	Terminée	↳ Valider un mot	Fabrice Valleix	29-04-2013 19:21
<input type="checkbox"/>	298	Module-#280: Jeu	Tâche	<div></div>	Terminée	↳ Retourner le nombre de points obtenus	Antoine de Roquemaurel	29-04-2013 19:21
<input type="checkbox"/>	281		Module	<div></div>	En cours	InterfaceTexte	Fabrice Valleix	29-04-2013 19:26
<input type="checkbox"/>	293	Module #281: InterfaceTexte	Tâche	<div></div>	En cours	↳ Permettre la saisie d'un mot	Fabrice Valleix	29-04-2013 19:21
<input type="checkbox"/>	294	Module #281: InterfaceTexte	Tâche	<div></div>	Terminée	↳ Afficher nombre de points	Fabrice Valleix	29-04-2013 19:24
<input type="checkbox"/>	303	Module #281: InterfaceTexte	Tâche	<div></div>	En cours	↳ Interface générale	Fabrice Valleix	29-04-2013 19:26
<input type="checkbox"/>	282		Module	<div></div>	En cours	InterfaceGraphique	Antoine de Roquemaurel	30-04-2013 18:36

FIGURE 4.1 – Affichage des demandes dans *Redmine*

Comme le montre la figure 4.1, nous avons la possibilité de lister toutes les tâches, nous pouvons voir à qui elles sont assignées, leurs dates limites, leur hiérarchie etc...

Redmine génère également un diagramme de Gantt, nous avons donc utilisé cet utilitaire pour nos diagrammes de Gantt¹ et la liste de nos tâches.

1. Disponible en annexe page 17

C'est également sur le wiki de *Redmine* qu'était disponible les conventions d'écritures, mais également des notes sur le contrôle de la qualité d'un code, et enfin le fonctionnement de *Git* et *Doxygen*.

4.2 Un logiciel de versionnement : Git

Afin de limiter les problèmes du travail collaboratif, nous avons utilisé un logiciel de versionnement Git. Il a deux intérêt, tout d'abord, nous pouvons travailler à deux en parallèle sur le projet sans se soucier de fusionner notre travail².

D'autre part, tous les logs étant enregistrés, nous pouvons savoir qui à fait quoi et quel jour, cela permet de voir également l'avancée du projet.

Enfin, toutes les modification sont stockées sur le serveur, ainsi en cas de problème, il est très facile de revenir à la version précédente ou même de comparer deux versions afin de voir les changements et de comprendre rapidement pourquoi une fonctionnalité a régressé.

2. À condition de ne pas travailler sur deux lignes de code identiques

Convention d'écriture en C

Voici les conventions écritures que nous avons fixé, il faudra les respecter pour que nous ayons un code propre et homogène, de plus elles ont été fixées pour que ce soit le plus simple pour nous (lecture rapide, propreté etc...)

A.1 Le nommage

A.1.1 Les variables globales

Les variables globales doivent être évitées. Utiliser une variable globale est une abomination, mais si cette utilisation est indispensable, celle-ci doit être préfixée par `g` comme ceci.

```
1 | int gMaVariable;
```

Listing A.1 – Exemples de noms de variable globale

A.1.2 Les variables en paramètre de méthodes

Les variables paramètre sont les variables qui ne sont que dans une méthode (et donc, elles sont détruites à la fin de la méthode). Ces variables doivent respecter la règle précédente à la différence qu'elles doivent toutes commencer par un `p_` (pour paramètre)

Ci un paramètre n'est jamais modifié durant la fonction, celui-ci doit être précédé du mot clef `const`.

```
1 | int pMonSuperParametre;  
2 | bool pVousAvezPerdu;  
3 | char* pCacamou;  
4 | const pMachinChose;
```

Listing A.2 – Exemples de noms de paramètres

A.1.3 Les noms de constantes ou `define`

Les constantes ou `#define` doivent être tout en majuscule, les différents mots de la constante sont séparés par des underscores (`_`). Même remarque que pour les attributs, choisissez des noms de constante clairs, compréhensibles par tous, pas seulement par ceux qui sont dans votre tête!

Il est préférable d'utiliser des constantes plutôt que des `define`, celles-ci ayant un typage fort contrairement à ces derniers.

```
1 | #define MA_SUPER_CONSTANTE;  
2 | #define VOUS_AVEZ_PERDU;  
3 | const int CONSTANTE;
```

Listing A.3 – Exemples de noms de constantes

A.1.4 Les noms de fonctions

Les fonctions doivent commencer par une minuscule, et séparer les différents mots par une majuscule. Les fonctions ne retournant rien (procédures) doivent toujours être à l'infinitif. À l'opposé les fonctions retournant quelques choses doivent être au participe passé. Les fonctions retournant un bouleen doivent être préfixé par le verbe afin d'avoir une lisibilité maximale. `estInferieur` par exemple Il faut décomposer au maximum, n'hésitez pas à faire une méthode `private` si besoin est, c'est toujours plus clair d'avoir une fonction, dictant explicitement ce qu'elle fait par son nom que 10 lignes de code bizarroïdes avec 2-3 lignes de commentaire! Et donc, les noms de fonctions sont essentiels!!

Chaque noms de fonction doit être préfixé par le nom du module suivis d'un underscore(_).

Ci une fonction ne contient aucun paramètre, celle-ci doit posséder le paramètre `void`.

```

1 void afficher(char* pTexteAAffiche){
2     printf("%s", pTexteAAffiche);
3 }
4 bool estInferieur(int entier1, int entier2){
5     return (entier1 < entier2);
6 }
7 void afficherTexte(void){
8     printf("coucou");
9 }
```

Listing A.4 – Exemple de fonctions

A.1.5 Les noms de type

Les noms de type doivent tous commencer par une majuscule, les différents mots sont séparés par une majuscule, choisissez des noms de types claires! (oui, je me répète, mais c'est ce qui fait toute la compréhension facile, ou non, d'un programme les noms de variables, classe, types, paramètre, méthodes etc...)

```

1 typedef struct {
2     int uneVariable;
3     char* uneAutreVariable;
4 } MonSuperType;
```

A.2 L'indentation

La règle est simple, on ouvre une accolade, la ligne suivante sera décalé vers la droite(une tab = 4 caractères), on ferme une accolade, on décale l'accolade vers la gauche et tout ce qui suis. Egalemt, si une ligne est trop longue, on va a la ligne, et décalons d'une ligne vers la droite, une fois l'instruction finie, on redécale vers la gauche. Dans le cas d'un switch, le break doit s'aligner avec le case 42 : tout ce qui est entre case et break sera indenté.

```

1 void afficherHelloWorld(){
2     printf("Hello World");
3     switch(yatta){
4         case 42:
5             // ^^
6             break;
7         case 1337:
8             // ...
9             break;
10        default:
11            //
12    }
13 }
```

14 | }

Listing A.5 – Exemple d'indentation

A.3 Les accolades












Les accolades ouvrante sont positionnés à la fin de la ligne demandant une accolade (switch, if, class, else, elseif, ...)
Les accolades fermantes sont positionnés une ligne après la dernière instruction. (avec une désindentation) Les else et elseif se mettent sur la même ligne que l'accolades fermante.

```
1      if(true){  
2          // bla bla  
3      } else if(false){  
4          // bla bla  
5      } else {  
6          //instruction  
7      }  
8  
9      switch(var){  
10         case 0:  
11             // ^^  
12             break;  
13         case 1:  
14             // yatta  
15             break;  
16         default:  
17             // :-)  
18     }  
19 }
```

Listing A.6 – Exemple d'indentation

Projet de Boggle

	2013-3				2013-4				2013-5				
	10	11	12	13	14	15	16	17	18	19	20	21	22
Cours	Cours												
L2													
S4													
Projet de Boggle	<div><div></div><div></div></div> Projet de Boggle												
Spécifications													
Conception													
Utile	<div><div></div><div></div></div> En cours 90%												
Plateau	<div><div></div></div> Terminée 100%												
Créer plateau	<div><div></div></div> Terminée 100%												
Générer grille de Boggle (...)	<div><div></div></div> Terminée 100%												
Dictionnaire	<div><div></div></div> Terminée 100%												
Créer dictionnaire	<div><div></div></div> Terminée 100%												
mot est dans dictionnaire	<div><div></div></div> Terminée 100%												
Obtenir probabilité d'apparition (...)	<div><div></div></div> Terminée 100%												
Resolveur	<div><div></div></div> Terminée 100%												
Résoudre une grille	<div><div></div></div> Terminée 100%												
mot est dans la grille	<div><div></div></div> Terminée 100%												
Nombre de mots commençant (...)	<div><div></div></div> Terminée 100%												
Jeu	<div><div></div></div> Terminée 100%												
lancer timer	<div><div></div></div> Terminée 100%												
Valider un mot	<div><div></div></div> Terminée 100%												
Retourner le nombre de (...)	<div><div></div></div> Terminée 100%												
InterfaceTexte	<div><div></div><div></div></div> En cours 67%												
Permettre la saisie d'un (...)	<div><div></div><div></div></div> En cours 60%												
Afficher nombre de points	<div><div></div></div> Terminée 100%												
Interface générale	<div><div></div><div></div></div> En cours 40%												
InterfaceGraphique	<div><div></div><div></div></div> En cours 80%												
Afficher solution	<div><div></div></div> Terminée 100%												

Afficher aide	 Nouveau 0%
Afficher nombre de points	 Terminée 100%
Sélectionner lettre pour (...)	 Terminée 100%
Interface générale	 Terminée 100%
Refactoring	 Nouveau 50%
Test utiles	 En cours 80%
Tests résolveur	 Nouveau 0%
Tests plateau	 Nouveau 0%
Tests dico	 Nouveau 0%
Tests jeu	 Nouveau 0%
Documentation	 Nouveau 80%
Ecrire solution dans fichier	
Résolveur à partir d'une case (...)	
Scanf vers fgets	
Gestion des arguments	
Tests fonctionnels	
Demande taille grille	
Grilles carrés par grilles (...)	

Université Toulouse III – Paul sabatier
L2 Informatique
Projet tuteuré

Antoine de ROQUEMAUREL
Fabrice VALLEIX
Groupe 2.2

Dossier de conception préliminaire (Mis à jour)

Projet de Boggle

Toulouse, le 10 mai 2013

Table des matières

1	But du document	3
2	Diagramme de décomposition en modules	3
3	Description des différents modules	4
3.1	Module Utile	4
3.2	Module Plateau	4
3.3	Module Dictionnaire	4
3.4	Module Resolveur	5
3.5	Module Jeu	5
3.6	Module InterfaceTexte	5
3.7	Module InterfaceGraphique	5
3.8	Module Solution	6
4	Répartition des tâches entre chaque membre	6
5	Calendrier de réalisation des tâches	8
6	Plan de tests	9
6.1	Plateau	9
6.2	Dictionnaire	9
6.3	Résolveur	9
6.4	Jeu	10
6.5	InterfaceGraphique et InterfaceTexte	10
6.6	Globalité	10
A	Annexes	11
A.1	Table des figures	11
A.2	Liste des tableaux	11

1 But du document

⚠ Ce document à été mis à jour par rapport au document initial, nous avons insérés dedans les modifications effectués a l'application par rapport à e qui a été prévu. La seul modification à été l'ajout d'un module **Solution** afin d'avoir une gestion facilitée.

C'est une description de haut niveau du produit, c'est-à-dire l'architecture générale du système, en termes de « modules », de sous modules et de leurs interactions. De plus, chaque module doit être décrit (définition des interfaces et des fonctionnalités générales). Ce document doit en premier lieu asseoir la confiance en la finalité et la faisabilité du produit, et, en second lieu, servir de base pour l'estimation des tâches à effectuer et du calendrier de leur réalisation.

Le « Dossier de Conception Préliminaire » doit également mettre en évidence le plan de tests, en termes de besoins de l'utilisateur, et montrer que l'on peut y satisfaire grâce à l'architecture proposée.

2 Diagramme de décomposition en modules

La description détaillée des différents modules est disponible section 3, la partie en rouge correspond au module que nous avons choisis d'ajouter.

R Afin de ne pas surcharger le schéma, le module **Utile** n'a pas été représenté ici, en effet tous les modules du projet sont susceptibles d'en avoir besoin, de plus ce module ne contient pas de fonctions spécifiques au projet mais des fonctions utiles travaillant sur des types de bases.

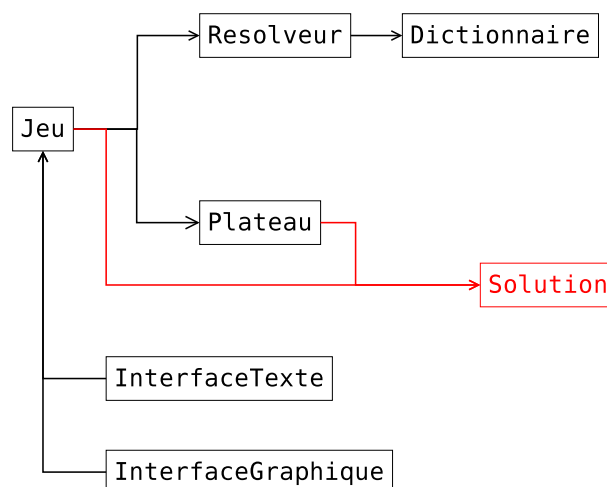


FIGURE 1 – Diagramme de décomposition en modules

3 Description des différents modules

Un diagramme représentant l'interaction entre les différents modules est disponible section 2.

3.1 Module Utile

Rôle	Toutes les fonctions de bases utiles au projet, ces fonctions travaillent sur des types de bases et ne sont pas spécifiques au projet, mais ce module permet de mieux organiser le code.
Type de données	Contient uniquement des traitements
Dépendances	Aucune
Fonctionnalités fournies	La liste sera complétée au fur et à mesure du projet en fonction des besoins nécessaires, en voici déjà quelques uns : supprimer les accents d'une chaîne de caractère, mettre une chaîne de caractère en majuscule, n'afficher un message qu'en cas de mode debug, trouver la première chaîne de caractère présente dans un tableau, retourner un booléen en fonction d'une certaine probabilité, etc...

TABLE 1 – Module Utile

3.2 Module Plateau

Rôle	Gérer la grille de Boggle
Type de données	Tableau à deux dimensions de <code>char</code>
Dépendances	Utile(3.1), Solution(3.8)
Fonctionnalités fournies	Générer la grille, Retourner la lettre concernant une case donnée

TABLE 2 – Module Plateau

3.3 Module Dictionnaire

Rôle	Gérer le dictionnaire du Boggle
Type de données	Fichier <code>FILE*</code> pointant sur le dictionnaire
Dépendances	Utile(3.1)
Fonctionnalités fournies	Initialiser le dictionnaire, parcourir le dictionnaire, dire si un mot est présent dans le dictionnaire ou non.

TABLE 3 – Module Dictionnaire

3.4 Module `Resolveur`

Rôle	Résoudre une grille de Boggle
Type de données	Structure contenant la grille de Boggle, le dictionnaire et un tableau de <code>char*</code> avec tous les mots possibles
Dépendances	<code>Dictionnaire(3.3)</code> , <code>Plateau(3.2)</code> , <code>Utile(3.1)</code>
Fonctionnalités fournies	Résoudre la grille, signaler si un mot est présent dans la grille, retourner la liste des mots de la grille commençant par une lettre.

TABLE 4 – Module `Resolveur`

3.5 Module `Jeu`

Rôle	Jouer au Boggle
Type de données	Structure contenant le Plateau et le Résolveur
Dépendances	<code>Plateau(3.2)</code> , <code>Resolveur(3.4)</code> , <code>Utile(3.1)</code> , <code>Solution(3.8)</code>
Fonctionnalités fournies	Proposer une lettre, Lancer le compte à rebours, Signaler si un mot proposé est correct, retourner le nombre de point obtenus, signaler si le joueur à gagner le jeu ou non

TABLE 5 – Module `Jeu`

3.6 Module `InterfaceTexte`

Rôle	Afficher et permettre de jouer au Boggle en mode texte
Type de données	<code>Jeu</code>
Dépendances	<code>Jeu(3.5)</code> , <code>Utile(3.1)</code>
Fonctionnalités fournies	Toutes les fonctions d’affichage et de saisie

TABLE 6 – Module `InterfaceTexte`

3.7 Module `InterfaceGraphique`

Rôle	Afficher et permettre de jouer au Boggle en mode semi graphique
Type de données	<code>Jeu(3.5)</code>
Dépendances	<code>Jeu</code> , bibliothèque externe <code>ncurses</code> , <code>Utile</code>
Fonctionnalités fournies	Toutes les fonctions d’affichage et de saisie

TABLE 7 – Module `InterfaceGraphique`

3.8 Module Solution

Rôle	Gère le stockage d'une solution
Type de données	structure contenant un <code>char**</code> et un <code>int</code>)
Dépendances	<code>Utile</code>
Fonctionnalités fournies	Ajout d'un mot dans la solution, Obtenir le nombre de mots, dire si un mot est présent ou non.

TABLE 8 – Module Solution

4 Répartition des tâches entre chaque membre

Un module sera toujours affecté à un membre du groupe, celui-ci sera en charge de vérifier que le module avance dans le temps impartis, et de s'occuper de l'intégration. Chacune des tâches seront affecté à un membre du groupe qui devra implémenter la tâche dans les délais prévus.

Le module `Utile` ne possède personne qui lui est assigné, en effet ce module se remplira en fonction de l'avancement des autres modules, et sera donc développé par les deux membres du binôme tout au long du projet.

L2 - Projet (Boggle) - Demandes

#	Tâche parente	Tracker	Début	Echéance	Sujet	Assigné à
L2 - Projet (Boggle) (28)						
250		Document	03-03-2013		Conception	
276		Module	16-03-2013	30-04-2013	Utilité	
277		Module	16-03-2013	23-03-2013	Plateau	Fabrice Valleix
284	Module #277: Plateau	Tache	16-03-2013	23-03-2013	Créer plateau	Antoine de Roquemaurel
285	Module #277: Plateau	Tache	19-03-2013	23-03-2013	Générer grille de Boggle en fonction de la taille	Fabrice Valleix
278		Module	16-03-2013	20-03-2013	Dictionnaire	Fabrice Valleix
286	Module #278: Dictionnaire	Tache	16-03-2013	17-03-2013	Créer dictionnaire	Antoine de Roquemaurel
287	Module #278: Dictionnaire	Tache	16-03-2013	20-03-2013	mot est dans dictionnaire	Antoine de Roquemaurel
288	Module #278: Dictionnaire	Tache	16-03-2013	19-03-2013	Obtenir probabilité d'apparition d'une lettre	Fabrice Valleix
279		Module	21-03-2013	10-04-2013	Resolveur	Antoine de Roquemaurel
299	Module #279: Resolveur	Tache	21-03-2013	05-04-2013	Résoudre une grille	Antoine de Roquemaurel
300	Module #279: Resolveur	Tache	05-04-2013	08-04-2013	mot est dans la grille	Fabrice Valleix
301	Module #279: Resolveur	Tache	06-04-2013	10-04-2013	Nombre de mots commençant par une séquence de lettre	Fabrice Valleix
280		Module	10-04-2013	16-04-2013	Jeu	Fabrice Valleix
290	Module #280: Jeu	Tache	10-04-2013	16-04-2013	lancer timer	Fabrice Valleix
297	Module #280: Jeu	Tache	10-04-2013	16-04-2013	Valider un mot	Fabrice Valleix
298	Module #280: Jeu	Tache	10-04-2013	14-04-2013	Retourner le nombre de points obtenus	Antoine de Roquemaurel
281		Module	16-04-2013	26-04-2013	InterfaceTexte	Fabrice Valleix
293	Module #281: InterfaceTexte	Tache	23-04-2013	26-04-2013	Permettre la saisie d'un mot	Fabrice Valleix
294	Module #281: InterfaceTexte	Tache	23-04-2013	26-04-2013	Afficher nombre de points	Fabrice Valleix
303	Module #281: InterfaceTexte	Tache	16-04-2013	23-04-2013	Interface générale	Fabrice Valleix
282		Module	16-04-2013	30-04-2013	InterfaceGraphique	Antoine de Roquemaurel
291	Module #282: InterfaceGraphique	Tache	23-04-2013	27-04-2013	Afficher solution	Antoine de Roquemaurel
292	Module #282: InterfaceGraphique	Tache	23-04-2013	26-04-2013	Afficher aide	Antoine de Roquemaurel
295	Module #282: InterfaceGraphique	Tache	23-04-2013	26-04-2013	Afficher nombre de points	Antoine de Roquemaurel
296	Module #282: InterfaceGraphique	Tache	23-04-2013	30-04-2013	Sélectionner lettre pour saisie	Antoine de Roquemaurel
302	Module #282: InterfaceGraphique	Tache	16-04-2013	23-04-2013	Interface générale	Antoine de Roquemaurel
283		Technical Story	19-04-2013	30-04-2013	Refactoring	Antoine de Roquemaurel

FIGURE 2 – Liste des tâches et leur répartition

5 Calendrier de réalisation des tâches

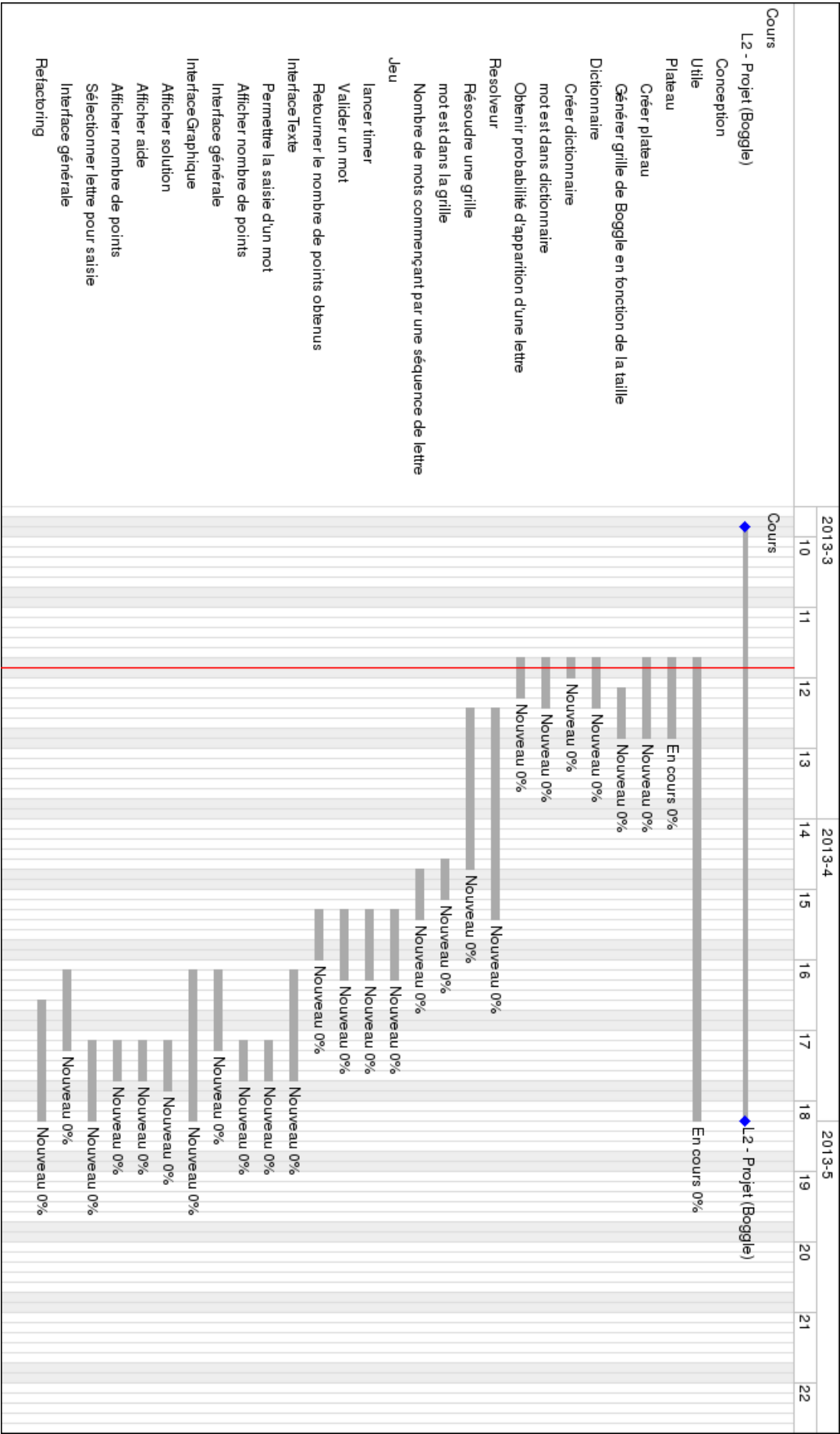


FIGURE 3 – Diagramme de Gantt

6 Plan de tests

Chaque fonction de chaque module sera testée à l'aide de tests unitaire, s'appuyant sur la bibliothèque `cunit`. Chacun des tests unitaire au pour but de tester un et un seul cas, mais l'ensemble des tests unitaires d'un module devront avoir passer en revue tous les cas possibles d'appel d'une fonction : Que ce soit un cas nominal, ou un cas d'erreur, ainsi chacune des lignes de code auront été testé, si ce n'est pas possible, du code mort aura été détecté.

Afin d'avoir des tests unitaires les plus efficaces possibles, ceux-ci ne seront pas développés par la personne qui a développé le module, ainsi cela permettra au deuxième membre du binôme de bien relire le code, éventuellement l'améliorer, et étant donné que ce n'est pas lui qui a développé le module, les fonctions de tests auront moins de risques d'être faussées.

Une fois que chaque fonction aura passé les tests unitaires, nous allons tester les modules séparément afin de vérifier que toutes les fonctions fonctionnent bien entre elle, ces tests seront différents en fonctions des modules.

6.1 Plateau

Un test fonctionnel aura lieu afin de tester le module, pour cela, nous appellerons la fonction de génération de grille et vérifierons qu'elle a bien généré une grille en fonction des tailles que nous lui donnons. Un jeu de grille sera généré, afin de vérifier que les lettres présentes dans la grille sont en fonction de leur utilisation dans la langue Française, un facteur de probabilité entrant en compte, il n'est pas possible d'attendre une réponse exacte.

Ce test s'effectuera tout d'abord dans son fonctionnement nominal, pour toutes les tailles que l'utilisateur est susceptible de rentrer (de 2 à 15, cette taille étant fixée dans les spécifications), ensuite un test s'effectuera sur des valeurs alternatives : nombre négatif, flottants, supérieur à 15 etc...et nous vérifierons que les erreurs sont bien gérés.

6.2 Dictionnaire

Afin de tester le dictionnaire, il faudra vérifier que la fonction permettant de savoir si un mot est présent dans le dictionnaire est correcte. Afin de tester cette fonction nous aurons un jeu d'essai comportant des mots présents, ou non dans la dictionnaire, au début du dictionnaire, à la fin, ou au milieu, des mots de longueur variable allant de 3 caractères, jusqu'à des mots de 20 caractères et vérifierons que les retour de fonctions sont bien ceux attendus.

6.3 Résolveur

Pour tester le Résolveur, nous essayerons avec des grilles prédéfinies de tailles variables, et vérifierons que le résolveur retourne bien tous les mots présent dans ces grilles, ceci sans erreurs. Le résolveur dépendant du dictionnaire, nous devons avoir testé préalablement le dictionnaire afin de pouvoir tester ce module.

Afin de vérifier tous les cas possibles, nous utiliserons le plus de grilles possible, tout d'abord des

grilles classiques de taille 4×4 , avec le plus de mots possibles présent dans la grille. Ensuite nous testerons le résolveur sur des grilles de taille 15×15 afin de vérifier qu'il n'est pas trop lent par rapport à ce que nous avons énoncé dans les spécifications. Également, nous lui ferons passer une grille où aucun mot n'est possible dans la grille et une grille de taille 2×2 afin de vérifier que dans ces cas extrêmes, le module fonctionne correctement.

6.4 Jeu

Afin de tester ce module, une interface est indispensable, ainsi nous nous en tiendrons aux tests unitaires, afin de vérifier que toutes les fonctions du module fonctionnent, ensuite ce module sera testé via l'intermédiaire de l'interface en mode texte.

6.5 InterfaceGraphique et InterfaceTexte

Ces deux modules sont les interfaces du jeu, il est difficile de faire des tests automatisés pour les interfaces, de plus ces deux modules sont intimement liés au contrôleur, `Jeu`, ainsi nous ne pourrions pas tester les interfaces séparément des autres modules, ces deux interfaces seront donc testées à l'aide d'un test fonctionnel lors d'une partie de Boggle. Ce test s'effectuera avec une grille prédéfinie.

6.6 Globalité

Une fois tous les tests effectués, nous testerons l'application complète, génération de la grille incluse, en spécifiant des tailles de grilles différentes, allant de 2 à 15.

A Annexes

A.1 Table des figures

1	Diagramme de décomposition en modules	3
2	Liste des tâches et leur répartition	7
3	Diagramme de Gantt	8

A.2 Liste des tableaux

1	Module <code>Utile</code>	4
2	Module <code>Plateau</code>	4
3	Module <code>Dictionnaire</code>	4
4	Module <code>Resolveur</code>	5
5	Module <code>Jeu</code>	5
6	Module <code>InterfaceTexte</code>	5
7	Module <code>InterfaceGraphique</code>	5
8	Module <code>Solution</code>	6