

L2 Science pour L'ingénieur

# Rapport de Projet

Réalisation d'un jeu type : WordBrain

Thomas Larmignat, Samed Oktay, Ouassim Messagier

# Projet WordBrain

Dans ce rapport nous allons tenter d'exposer la façon dont nous avons abordé notre sujet : WordBrain. La réussite de ce projet nécessite une bonne répartition du travail dans mais aussi une bonne utilisation des outils qui une meilleure optimisation du travail mais une meilleure coordination dans le travail d'équipe.

Dans notre projet nous avons tenté d'appliquer la façon dont nous avons traité le sujet en trois grand axes : Une analyse commune du sujet et de ces exigences, une repartitions des grands problèmes du sujet puis une phase de test et des débogages communes, nous allons présenter notre sujet selon le plan suivant :

## Sommaire :

### I. Organisation et analyse du sujet

I.1 Règles du jeu et analyse du sujet .....	p.2
I.2 Cahier des charges.....	p.2
I.3 Organisation et répartitions des charges .....	p.2-3

### II. Développement et outils de développement

II.2 Déroulement du développement et difficultés rencontrées .....	p.4
II.3 Utilisation d'outils de développement .....	p.10

### III. Résultats et conclusions

III.1 Résultats et objectifs atteints .....	p.15
III.2 Conclusion .....	p.15

## I. Organisation et analyse du sujet

Le but du jeu WordBrain est de trouver des mots cachés dans une grille de lettres, quand un mot est trouvé, il faut glisser son doigt sur l'écran tactile pour le former, les mots peuvent être formés vers le haut, le bas, à gauche, à droite mais aussi en diagonale. Il faut trouver des mots en particulier pour résoudre une grille de WordBrain, la taille des mots est indiquée. L'ordre dans lequel sont trouvés les mots est important, en effet comme sur l'exemple ci-contre, on a deux mots à trouver mais on peut former le mots « AIDE » en premier. Une fois formé, il va disparaître de la grille, et les lettres « DE » du mot « GARDE » vont descendre, et le mot « GARDE » va pouvoir être formé.

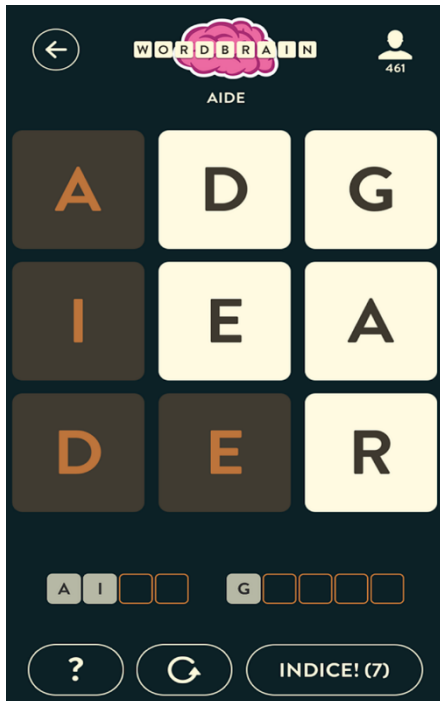


Figure 1 : Grille du jeu WordBrain®

A partir de ces règles, nous avons tenté d'écrire un cahier des charges qui répond aux mieux aux exigences et aux règles du sujet :

- Un programme qui reprend les principales fonctionnalités du jeu :
  - Grilles de mots connexes générées aléatoirement avec différentes listes de mots (4 lettres, 5 lettres ...).
  - Lettres qui tombent lorsque qu'un mot est trouvé.
  - Un mode qui permet de jouer sur différents niveaux, avec des niveaux composés de grilles de différentes tailles.
- Un programme qui répond aux exigences du projet :
  - Un système de sauvegarde et chargement à partir d'un fichier.
  - Une interface qui doit être adaptée à un terminal

Pour pouvoir répondre à ce cahier des charges, dès le début de notre projet nous avons mis en place un planning qui a été modifié en fonction du travail effectué et du travail restant,

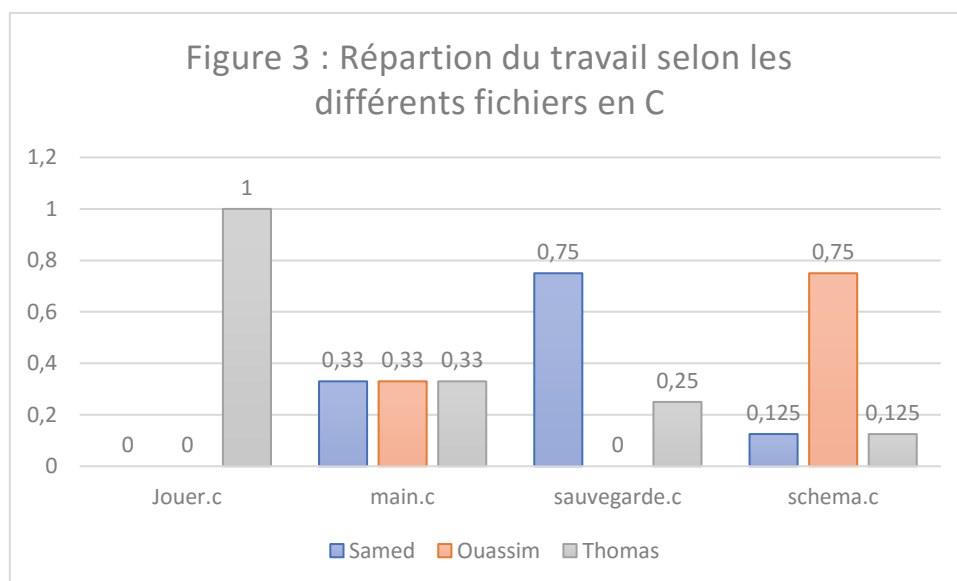
cela nous a permis une meilleure organisation dans notre travail avec un meilleure répartition des tâches.

On peut voir la répartition du travail de différentes manières, si on regarde le planning et le graphique ci-dessous, on remarque que les fichiers en C donnent une autre répartition du travail que le planning, mais cette répartition est faussée par le nombre de ligne de chaque

NOVEMBRE 2016																														
semaine	44	44	44	44	44	44	45	45	45	45	45	45	45	46	46	46	46	46	46	47	47	47	47	47	47	47	48	48	48	
	M	M	J	V	S	D	L	M	M	J	V	S	D	L	M	M	J	V	S	D	L	M	M	J	V	S	D	L	M	M
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
Wassim	Mise en place des règles							schémas + trouver les mot							mots qui tombent										Déboggage					
Thomas	et							Listes de mots+schémas							Mots Aléatoires + génération grilles							Mode de jeu avec nvx					sauvergarde			
Samed	du cahier des charges							schémas							Génération grilles							Déboggage					Indices			
DECEMBRE 2016																														
semaine	48	48	48	48	49	49	49	49	49	49	50	50	50	50	50															
	J	V	S	D	L	M	M	J	V	S	D	L	M	M	J	V														
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16														
Wassim	Indices				Déboggage				Compte rendu																					
Thomas	Interfaces				Déboggage				Compte rendu																					
Samed	sauvergarde + Doxygen				Reset				Compte rendu																					

Figure 2 : Planning avec réparation des tâches au sein du groupe

fichier.



L'organisation et la gestion du temps sont deux choses essentielles pour mener un projet à bien, nous avons rencontré quelques difficultés pour la mise en place d'un planning prédéfini mais ceci nous a semblé indispensable pour une meilleure cohésion et communication dans le groupe. Nous allons maintenant parler de la partie la plus importante du projet, le développement.

## Développement et outils de développement

Dès le début de notre projet, nous avons divisé notre sujet en plusieurs grands problèmes :

- La génération de grilles de mots connexes aléatoires.
- Comparaison d'une réponse par rapport à la solution.
- Faire « tomber » les lettres lorsqu'un mot est trouvé.
- Le système de sauvegarde.
- Petites fonctionnalités (indices, interfaces ...).

Nous avons tenté de répondre à tous ces problèmes pour répondre au mieux au cahier des charges pour avoir un programme très proche de celui imposé par ce cahier :

Il nous faut générer des grilles aléatoires, c'est-à-dire mettre des mots de tailles prédéfinies, qui seront eux même sélectionnés aléatoirement dans le fichier contenant des mots de la taille rechercher.

```
void schema3_1(int mat[10][10],char mot3[20]){
    int a;
    mat[0][0]=0;
    mat[0][1]=1;
    mat[0][2]=2;
    mat[1][2]=3;
    mat[1][1]=4;
    mat[1][0]=5;
    mat[2][0]=6;
    mat[2][1]=7;
    mat[2][2]=8;
    int i;
    int j;
    for(i=0;i<3;i++){
        printf("  _  _  _ \n");
        printf("|  |  |  |  | \n");
        for(j=0;j<3;j++){
            a=mat[i][j];
            printf(" | %c | ",mot3[a]);
        }
        printf("\n|_| |_| |_| \n");
    }
}
```

*mise en place du schéma*

La deuxième difficulté de cette partie est que les mots doivent être connexes, pour cela nous avons donc décidé de mettre en place différents schémas pour les différentes tailles de grilles, pour cela on concatène tous les mots choisis aléatoirement dans une chaîne de caractère de la taille de la grille qui elle-même sera choisie aléatoirement parmi 8 schéma par taille de grille (3x3,4x4 ...).

Figure 3 : Exemple de schéma implémenté dans le code pour une matrice 3x3.

Par exemple pour une grille 3x3, prenons deux mots, **ROUGE** et **BLEU**, on concatène les deux mots **BLEUROUGE**, on place la chaîne dans un des schémas prédéfinis :

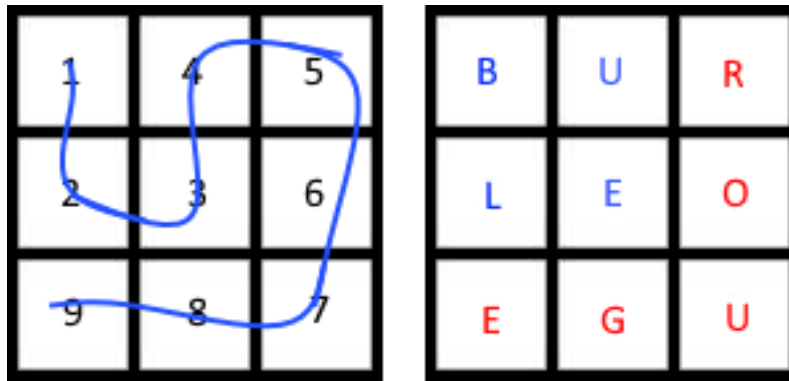


Figure 4 : A gauche "fil" placé que doit suivre la chaîne et à droite le résultat dans la matrice.

Nous avons donc décidé d'avoir une génération de grille qui n'est pas complètement aléatoire mais qui en donne l'illusion, cela nous a permis de faire des générations beaucoup plus naturelles (moins difficiles) mais avec des niveaux qui sont toujours aléatoires.

Pour répondre aux objectifs définis précédemment, il faut afficher la grille, comparer si sa réponse est correcte et faire tomber les lettres si la case du dessous est vide :

Chaque niveau de jeu est basé sur la fonction `gameniveau_` (fini par un chiffre en fonction de la taille de la grille du niveau). Dans un premier temps on récupère le nombre de mots avec une certaine taille de façon à ce que lorsqu'on concatène tous les mots, on doit obtenir le même nombre de lettres et le même nombre de cases de la grille.

Ensuite l'algorithme va choisir aléatoirement le schéma parmi ceux qui ont été définis pour cette taille de grille. On affiche la grille à l'utilisateur. Un système d'indices a également été implémenté, le nombre d'indices restant à l'utilisateur est également affiché.

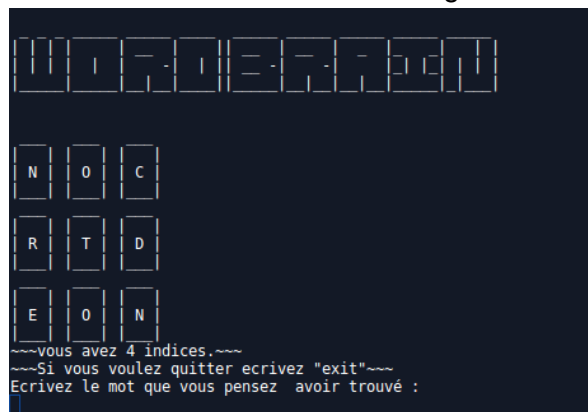


Figure 5 : Affichage de la grille et du nombre indices

La deuxième partie de la fonction, on récupère dans une chaîne de caractère la réponse proposée par l'utilisateur, on compare cette chaîne aux mots présents dans la grille, si le mot est présent il est retiré de la grille. Pour le retirer un mot de la grille, on cherche ce mot dans la chaîne concaténée et l'algorithme remplace les lettres du mot trouvé par des espaces.

Ensuite, il faut faire « tomber » les lettres s'il le faut (si la case en dessous la lettre est vide). Pour valider si l'utilisateur a fini une grille, il faut juste tester si la chaîne concaténée ne contient que des espaces.

```

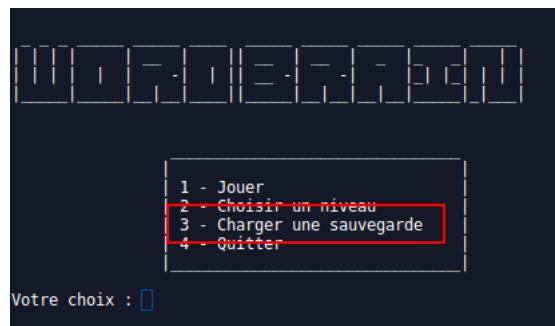
int bsansblanc(char *R){
    int i=0;
    int bAvcCaractere=0;
    int bSansCaractere=1;
    nChaineLg(R);
    while(R[i] != '\0'){

        if(R[i] != ' '){
            return bAvcCaractere;
        }
        i++;
    }
    return bSansCaractere;
}

```

**Figure 6 : bsansblanc(char \*R) est la fonction qui vérifie si la chaine concaténée ne contient que des espaces.**

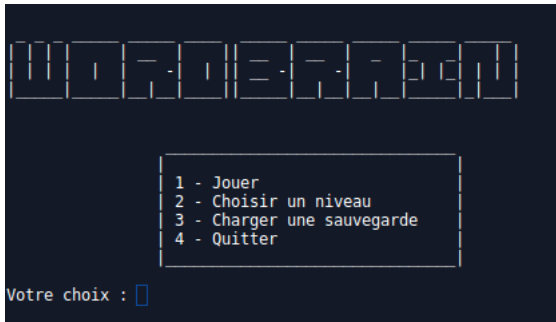
D'autres fonctionnalités, comme écrire « save » pour sauvegarder la grille ou encore, pouvoir réinitialiser le niveau ? Il faut écrire « reset », une autre commande de débogage est « », elle permet d'afficher les mots présents dans la grille sans avoir à les trouver .



Pour charger une sauvegarde, il suffit d'aller dans le choix 3 du menu principal. La sauvegarde n'est pas conservée une fois le programme quitté.

Le jeu WordBrain a été conçu pour les écrans tactiles, nous avons donc dû adapter la jouabilité du jeu à un terminal :

L'interface se présente sous la forme d'un menu basique en langage c, mais suffit largement en vue du temps et de la complexité du projet. Les menus se composent sous forme de switch :



Ci-contre, le menu principal du programme, il faut entrer son choix pour accéder à une fonctionnalité (1 pour Jouer en mode continu, 2 pour choisir un niveau indépendant, 3 pour charger une sauvegarde).

Figure 7 : Affichage du menu principal.



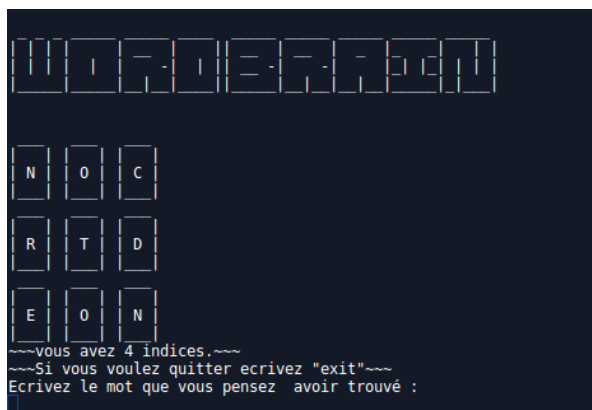
Si on tape 1 par exemple, on accède au menu qui nous permet de choisir le niveau de difficulté sur le mode de jeu continu.

On peut voir ci-contre, que tous les menus du jeu sont basés sur le même principe de menus avec un switch.

Figure 8 : Affichage menu de jeu continu, l'utilisateur choisit son niveau.

L'affichage des grilles est géré dans les fonctions schémas, l'affichage se fait par une boucle for qui répète l'affichage autant de fois qu'il y a de lignes en affichant une lettre dans chacune des cases affichées. Une fois un mot trouvé, il peut être écrit en minuscule ou en majuscule. En tapant « save », on sauvegarde la grille tant que l'on ne quitte pas le programme et si on tape « exit » on revient au menu principal, ces deux fonctionnalités seront détaillées plus tard dans le rapport.

S





```
for(i=0;i<3;i++){
    printf("  _  _  _ \n");
    printf("|  |  |  | \n");
    for(j=0;j<3;j++){
        a=mat[i][j];
        printf("| %c | ",mot3[a]);
    }
    printf("\n|__| |__| |__|\n");
}
```

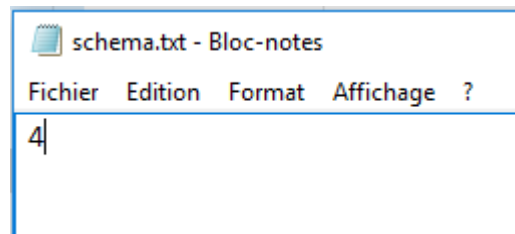
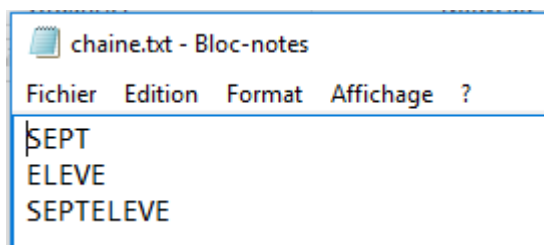


Une des fonctionnalités imposées par le cahier des charges est la sauvegarde et la possibilité de charger une sauvegarde pour continuer une partie. Nous avons mis très peu de temps à comprendre que le principe de la sauvegarde était simple : il suffit d'inscrire les données nécessaires dans un fichiers puis d'extraire ces informations pour pouvoir recharger une grille.

Figure 10 : Affichage d'un niveau et la fonction qui permet l'affichage des cases de la grille.

 schema.txt  
 chaine.txt

Nous avons donc mis en place un ensemble de fonctions permettant d'écrire dans notre premier fichier "chaine.txt" les mots et la concaténation de ces mots en dernière ligne du fichier. Nous pouvions avoir minimum 1 mot et maximum 5 mots donc nous devons compter le nombre de ligne inscrite dans ce fichier afin de pouvoir savoir le nombre de mot et à quelle ligne se trouve la chaîne concaténée. Une deuxième donnée nécessaire était le numéro du schéma utilisé durant la partie sauvegarder c'est pourquoi nous avons aussi créé une fonction qui écrit dans un deuxième fichier "schema.txt" le numéro du schéma. Maintenant que la partie sauvegarde est effectué il fallait pouvoir lire ces fichiers afin de pouvoir charger cette même partie. Nous avons donc créé plusieurs fonctions qui renvoies le premier mot, le deuxième, ... , la chaîne concaténée et le numéro du schéma.



Le principal problème pour notre système de sauvegarde est de savoir quel type de grille à été sauvegardé, donc cela revient à dire combien de mots sont présents dans le fichier chaine.txt sans compter la chaîne concaténée ? On peut résoudre ce problème par l'intermédiaire d'une autre fonction :

```

void sauvegarde(int mat[10][10],char *mot4,char *mot5,char *mot6,char *mot7,char *mot8,char *R) {
    int schema=lecture_schema();
    int nb=0;
    int ligne=0;
    nb=nombre_ligne(ligne);
    printf("%i \n",nb);
    int lg=0;
    R=lecture_R(R,nb);
    while(R[lg]!='\0'){
        lg++;
    }
    printf("%i \n",lg);
    switch(lg){
        case 4: save_niveau2(mat,mot4,R);break;
        case 9: save_niveau3(mat,mot4,mot5,R);break;
        case 16:save_niveau4(mat,mot4,mot5,mot6,R);break;
        case 25:save_niveau5(mat,mot4,mot5,mot6,mot7,mot8,R);break;
    }
    remove("chaine.txt");
}

```

Figure 11 : Fonction de sauvegarde en fonction de la longueur de la chaîne concaténée.

Cette fonction ouvre le fichier, lis la dernière ligne sachant qu'à celle-ci se trouve la chaîne concaténée. La chaîne comporte autant de caractère que la grille de case. Par exemple pour une grille de 3x3 la chaîne possède 9 caractères car il y a 9 cases. Ensuite il suffit d'un switch avec les valeurs 4,9,16 et 25 qui représentent les grilles 2x2,3x3,4x4 et 5x5.

Enfin nous avons créé une fonction pour chacune des grilles qui va donc initialiser la grille avec le(s) mot(s) en lisant le numéro du schéma utilisé dans le fichier "schema.txt". Pour terminer il suffit d'appeler nos différentes fonctions qui permettent de pouvoir jouer.

Durant l'ensemble du projet nous avons utilisé un certain nombre d'outil de développement qui nous ont aidé dans le développement aussi dans la commutation au sein du groupe :

Afin de déboguer notre programme nous avons utilisé lldb, nous l'avons utilisé de la façon suivante :

On place 3 breakpoints dans notre fonction gameNiveau3 pour effectuer un débogage :

The screenshot shows a code editor with the following C code for the `game_niveau3` function:

```

1951 * [param stage Progression lorsqu'un niveau de difficulté
est choisi
1952 */
1953
1954 void game_niveau3(int mat[10][10],char *mot4, char
*mot5,char *R,int continue, int niveau,int stage)
{
    //3X3
1955     int bEspace=0;
1956     int t=0;
1957     int compteur=0;
1958     int resultat=0;
1959     int aide=0;
1960     int saut=0;
1961     int quitter=0;
1962     int triche=0;
1963     int reset=0;
1964     char reponse[30];
1965     char *premier = fonc_mot4(mot4);
1966     char *deuxieme = fonc_mot5(mot5);
1967     char *mot6=" ";

```

On the right, the lldb terminal shows the following commands and output:

```

wassim@wassim-LIFEBOOK-A512:~/Bureau/Licence_SPI2_2016 $ lldb
(lldb) target create "monprog"
Current executable set to 'monprog' (x86_64).
(lldb) b schema.c: 1954
Breakpoint 1: where = monprog`game_niveau3 + 68 at schema.c:1955, address = 0x0000000040cc3d
(lldb)

```

A red arrow points from the lldb terminal to the function definition in the code.

```

2067         nb();
2068         invalide, désolé !\n");
2069     }
2070 }
2071     else{
2072         clear;
2073         wb();
2074         printf("Le mot est
2075 présent !\n");
2076 R=TrouverLeMotNiveau2(reponse,mat,premier,deuxieme,3);
2077         bEspace=bsansblanc
2078 (R);
2079         while(i!=5){
2080             t("Désolé vous n'avez plus
2081 d'indices");
2082             }
2083             if(aide==1 && compteur<4){
2084                 compteur++;
2085                 clear;
2086                 wb();
2087                 indice
2088 (R,premier,deuxieme,mot6,mot6,compteur);
2089             }
2090             if(triche==1){
2091                 clear;

```

```

wassim@wassim-LIFEBOOK-A512:~/Bureau/Licence_SPI2_2016 $ lldb monprog(lldb) t
target create "monprog"
Current executable set to 'monprog' (x86_64).
(lldb) b schema.c: 1954
Breakpoint 1: where = monprog`game_niveau3 + 68 at schema.c:1955, address = 0
x000000000040cc3d
(lldb) b schema.c: 2037
Breakpoint 2: where = monprog`game_niveau3 + 1198 at schema.c:2037, address =
0x000000000040d0a7
(lldb) b schema.c: 2075
Breakpoint 3: where = monprog`game_niveau3 + 1647 at schema.c:2075, address =
0x000000000040d268
(lldb)

```

Le programme s'arrête dès qu'on demande la fonction "gameniveau3" car ça correspond au breakpoint1 :

```

1952 //
1953 */
1954 void game_niveau3(int mat[10][10],char *mot4, char
*not5,char *R,int continu,int niveau,int stage)
{
1955     int bEspace=0;
1956     int l=0;
1957     int compteur=0;
1958     int resultat=0;
1959     int aide=0;
1960     int sauve=0;
1961     int quitter=0;
1962     int triche=0;
1963     int reset=0;
1964     char reponse[20];
1965     char *premier = fonce_mot4(mot4);
1966     char *deuxieme = fonce_not5(not5);
1967     char *mot6=" ";
1968     char sauvegarde[20]="SAVE";
1969     char indice[10]="INDICE";
1970     char *exit="EXIT";
1971     char *ishit="ISHIT";
1972     char *reset="RESET";
1973
1974     R=sc2(premier,deuxieme);
1975     int hasard = rand() % 4 + 1; //entre 1 &
1976     stockage_premier(premier);
1977     stockage_deuxieme(deuxieme);
1978     stockage_R(R);
1979     do{
1980         l++;
1981         premier=lecture_premier(premier,l);
1982         deuxieme=lecture_deuxieme(deuxieme,l);
1983         R=lecture_R(R,l);
1984         clear;
1985         wb();
1986         if((continu==1)&&(niveau==1)){
1987             printf("Niveau facile: %i/10

```

```

Process 4225 stopped
* thread #1: tid = 4225, 0x000000000040cc3d monprog`game_niveau3(mat=0x00007f
ffffffda90, mot4="", mot5="", R="", continu=0, niveau=0, stage=0) + 68 at sch
ema.c:1955, name = 'monprog', stop reason = breakpoint 1:1
frame #0: 0x000000000040cc3d monprog`game_niveau3(mat=0x00007ffffffda90,
mot4="", mot5="", R="", continu=0, niveau=0, stage=0) + 68 at schema.c:1955
1952 */
1953
-> 1954 void game_niveau3(int mat[10][10],char *mot4, char *not5,char *R,int
continu,int niveau,int stage){
1955     int bEspace=0;
1956     int l=0;
1957     int compteur=0;
1958     int resultat=0;
(lldb) cont

```

Ensuite le programme continue normalement lorsqu'on tape la commande "count" jusqu'à ce qu'on demande l'indice et il va effectuer le débogage qui correspond au breakpoint 2 (placé à la ligne 2037) :

```

1023     alde=bChaineEgale(reponse,indice);
1024     quitter=bChaineEgale(reponse,exit);
1025     triche=bChaineEgale(reponse,ishit);
1026     reset=bChaineEgale(reponse,RESET);
1027     if(compteur==0){
1028         clear;
1029         wb();
1030         t("Désolé vous n'avez plu
1031 d'indices");
1032     }
1033     if(alde==1 && compteur<1){
1034         compteur++;
1035         clear;
1036         wb();
1037         indice
1038         (R,premier,deuxieme,mot6,mot6,compteur);
1039     }
1040     if(triche==1){
1041         clear;
1042         wb();
1043         printf("mot1:ks .
1044 \n",premier);
1045         printf("mot2:ks .
1046 \n",deuxieme);
1047     }
1048     if(quitter==1){
1049         break;
1050     }
1051     if(alde==1 && triche==0){
1052         resultat =
1053         bChainesEgalesCompareMotsAvecSAutres
1054         (reponse,premier,deuxieme,mot6,mot6,mot6);
1055         if (resultat == 0){
1056             sauv = bChaineEgale
1057             (reponse, sauvegarde); //Debut récupération sauvegarde

```

Ensuite on continue le programme et lorsque l'utilisateur saisit le bon mot il rentre dans le else et on le voit bien avec ce qui est entouré en vert sur la photo et juste après le lldb effectue le débogage de notre dernier breakpoint3, on peut voir les détails sur les images :

```

1061     wb();
1062     //fin récupération
1063     sauvegarde
1064     }
1065     else{
1066         clear;
1067         wb();
1068         printf("Mot
1069 invalide, désolé !\n");
1070     }
1071     else{
1072         clear;
1073         wb();
1074         printf("Le mot est
1075 présent !\n");
1076         R=TrouverLeMotNiveau2(reponse,mat,premier,deuxieme,R);
1077         bEspace=bsansblanc
1078         (R);
1079         while(!l==5){
1080             R=tomberNiveau2(mat,R);
1081             i++;
1082         }
1083     }
1084     while(bEspace!=1 && reset!=1);
1085     clear;
1086     if(quitter==1){
1087         printf("\n\n\n\n\n\n\n\n");
1088         printf("
1089 la grille--\n");
1090         sleep(1);
1091         clear;

```

Et enfin lorsqu'on quitte le programme voici ce qu'on obtient (aucune erreur ou autres).

```

wassim@wassim-LIFEB00K-A512: ~/Bureau/Licence_SPI2_2016
Au revoir !
Programme terminé.
Process 4225 exited with status = 0 (0x00000000)
(lldb)

```

On utilise également la fonction `assert()` dans notre débogage dans ce cas, par exemple, on regarde dans la fonction `bChaineEgale` si l'entier `bEgale` comporte bien une valeur booléenne.

```

int bChaineEgale(char *sTexte1, char *sTexte2){ //strcmp(sTexte
    int nC;
    int bEgale = nChaineLg(sTexte1) == nChaineLg(sTexte2);
    if (bEgale) {
        for (nC = 0; nC < nChaineLg(sTexte1); nC++)
            bEgale = bEgale && sTexte1[nC] == sTexte2[nC];
    }
    Assert1("bChaineEgale", bBool(bEgale));
    return bEgale;
} //bChaineEgale

```

Figure 11 : Exemple d'assertion sur une fonction.

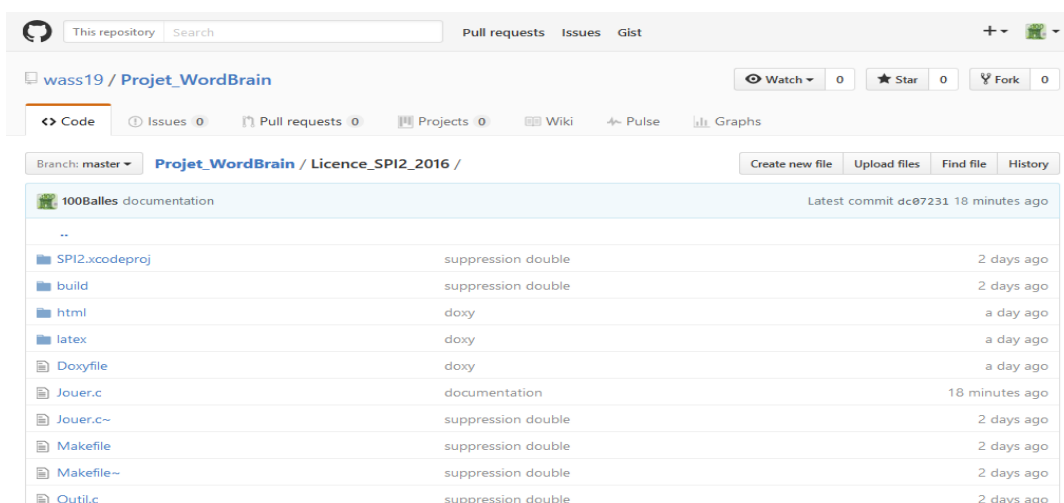
D'autres outils de développement autres que ceux de débogage nous ont été utiles durant ce projet :

Pour faciliter la communication au sein du groupe et pouvoir travailler sans risque de perte de données et partager sa progression au sein du groupe nous avons utilisé GitHub. :

Nous avons tout d'abord appréhendé le fait d'utiliser GitHub, mais une fois pris en main, son utilisation semble indispensable pour mener un projet de groupe à terme. Son utilisation est très simple et très pratique. En effet l'élément qui le différencie de tous les autres sites d'hébergement en ligne et le fait de pouvoir l'utiliser directement à partir de lignes de commandes sur le terminal. En utilisant GitHub on s'est rapidement rendu compte que le nombre de commandes à connaître peut être limité à 3.

Tout d'abord git pull qui permet de récupérer les modifications apportées par les autres membres du groupe et donc d'avoir un projet à jour. Ensuite si des modifications sont effectuées et que nous voulons les ajouter sur GitHub afin que les autres membres du groupe possèdent un dossier à jour, il suffit d'ajouter les fichiers modifiés avec git add <fichier>. Il faut ajouter à celui-ci un « commentaire » qui va permettre de décrire les modifications effectuées. Il suffit de faire git commit -m « commentaire ». Enfin il suffit d'envoyer sur le GitHub en utilisant la commande git push origin master.

Voici le lien pour accéder à notre dossier : [https://github.com/wass19/Projet\\_WordBrain/](https://github.com/wass19/Projet_WordBrain/)





Un autre point essentiel du projet est une documentation technique de notre programme, pour faire une documentation claire et compréhensible par tout le monde nous avons utilisé Doxygen qui est capable de générer une documentation de code possédant des capacités de génération de documentation à partir du code source d'un programme.

#### main.c File Reference

Wordbrain. [More...](#)

```
#include "Outil.h"
#include "Tas.h"
#include "schema.h"
```

Include dependency graph for main.c:

#### Macros

```
#define clear printf("e[1;1Hie[2J");
```

#### Functions

```
void main2 ()
void ModuleAmorceR ()
int main (int argc, const char *argv[])
```

#### Detailed Description

Wordbrain.

##### Author

OKTAY Samed / LARMIGNAT Thomas / MESSAGIER Thomas

##### Version

2.0

Le principe de l'autodocumentation est très simple : il suffit de commenter son code en utilisant différentes balises chacune spécifique à un objet. Comme dans l'exemple ci-dessous, il y a tout d'abord la balise `\file` qui permet de définir le nom du fichier qui est commenté. Nous pouvons aussi voir différentes balises comme `\brief`, `\author` et autres... où nous décrivons les différentes caractéristiques du fichier. C'est ici où l'on définit l'auteur, la fonction du fichier, la version du fichier et la date de création ou de modification de celle-ci. Ce premier bloc ne concerne que le fichier. Nous pouvons observer juste en dessous un bloc qui va décrire la fonction `jouer_facile`. Le bloc de documentation est composé de différentes balises:

- La première balise `\fn` où apparaît le nom de la fonction et de ses paramètres.
  - La deuxième balise `\brief` est une courte description de la fonction.
  - La troisième balise `\param` permet d'expliquer un paramètre et peut être appelée plusieurs fois à la suite comme nous pouvons le voir ici.
- Une autre balise peut aussi être utilisée : `\return` qui permet d'expliquer le résultat que retourne la fonction.

```

/**
 * \file Jouer.c
 * \brief Jouer de façon continu
 * \author OKTAY Samed / MESSAGIER Ouassim / LARMIGNAT Thomas
 * \version 2.0
 * \date 16 decembre 2016
 */
/**
 * \fn void jouer_facile(int mat[10][10],char *mot4, char *mot5,char *R,int continu)
 * \brief niveau de difficulté facile 4 grilles de 2x2 et 6 grilles de 3x3
 *
 * \param mat[10][10] Matrice de 10x10 contenant le(s) mot(s)
 * \param mot4 Premier mot
 * \param mot5 Deuxieme mot
 * \param R Concatenation des deux mots
 * \param continu Vaut 1 si cette fonction est appelé depuis le mode continu
 */
void jouer_facile(int mat[10][10],char *mot4, char *mot5,char *R,int continu){//niveau de difficulté facile 4 grilles de 2x2 et 6 grilles de 3x3///
    int i;
    int stage=1;
    int niveau=1;
    for(i=1;i<=4;i++){//4 grilles 2x2//
        clear;
        wb();
    }
}

```

Figure 12 : Exemple de bloc de documentation Doxygen.

Sur l'ensemble du projet, la plupart des objectifs initiaux ont été respecté, notre programme génère des grilles de mots connexes aléatoires, procède une possibilité de sauvegarder une grille, au début du projet, nous avons discuté sur la conception d'une interface graphique mais en vue de la complexité du sujet (qui peut sembler simple au premier abord) et peut-être un manque d'organisation vers la fin du projet, nous avons préféré garder une interface sur le terminal. Au tout début de notre projet nous avons conçu un planning prévisionnel. Ce planning a été en globalité suivi même quelques difficultés ont été rencontrer en fin de projet avec la sauvegarde et le débogage qui ont pris beaucoup de retard à cause de certaines difficulté (beaucoup de fonction à retravailler complètement), nous n'avons pas prévu cela dans notre planning. Notre programme pourrait être améliorer en y ajoutant, comme nous voulions le faire au début du projet, une interface graphique pour que les lettres soit cliquable et ainsi améliorer l'expérience de jeu.

Notre groupe est unanime, le projet nous a apporter une expérience supplémentaire dans les travaux de groupe, durant ce projet nous avons appris gérer notre temps et prendre des habitudes de travail en information : L'utilisation de GitHub est presque devenue une habitude, on a amélioré notre façon de commenté du code (Doxygen), mais aussi des habitudes de test, débogage. Pour nous ce projet est principalement centrer sur la matière algorithmique avancée et outils de développement qui nous a donné les bases du développement et le projet a perfectionné ces bases.