Mezouar Mikael Université Gustave Eiffel Dos Santos Loic

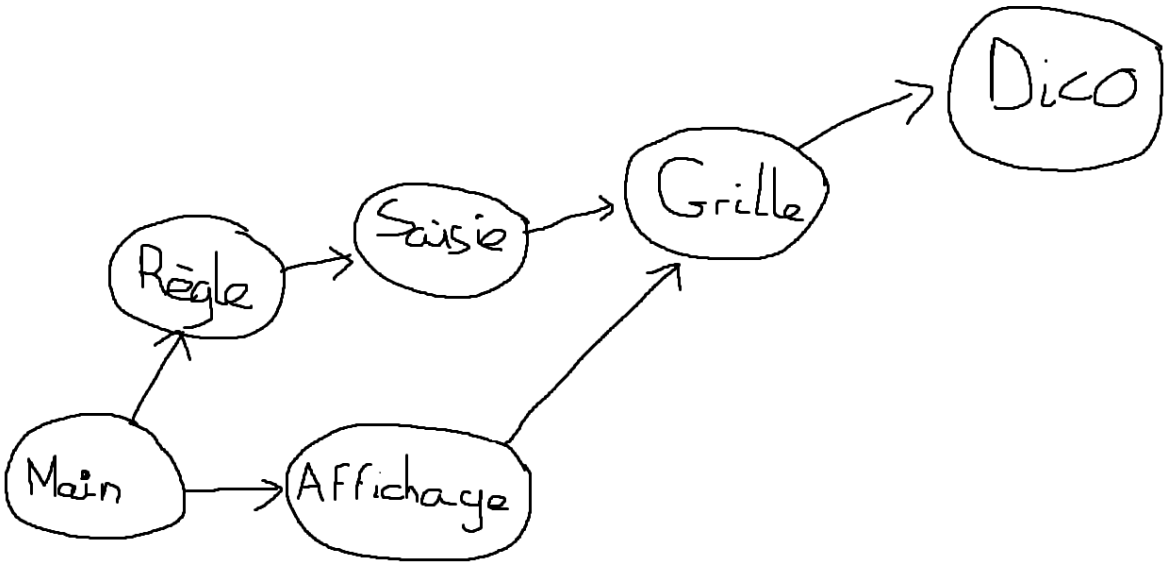
Rapport TP7 : Perfectionnement en programmation C

23/03/2021 L2 informatique

Introduction :

L'objectif de ce TP était de réaliser un jeu se nommant le Boggle , le principe du jeu est plutôt simple , une grille de lettre s'affiche et vous devez trouver un maximum de mots en formant des chaînes de lettres contiguës. Plus le mot est long, plus les points qu'il vous rapporte sont importants. Une lettre ne peut pas être utilisée plus d'une fois pour un même mot ,seuls les mots de trois lettres ou plus comptent et les accents ne sont pas importants. Pour notre jeu , la taille de la grille est de 4 . Ce TP est beaucoup moins directif que les précédents. Il ressemble plus à une véritable spécification fournie par un client. Nous allons voir dans ce rapport comment nous avons implémenté ce programme .

Modularisation :



Au niveau de la modularisation nous avons décidé de séparer notre programme en 6 modules :

-le module Dico ou nous avons implémenté un arbre lexicographique permettant par la suite de pouvoir stocker tout le mot du dictionnaire et les mots trouvés pendant le jeu dans 2 arbres distincts.

-le module Règle ou nous avons implémenté les fonctions qui s'occupe de gérer les règles du jeu .

-le module Affichage qui gère la partie graphique du jeu avec le module Ncurses .

-le module Grille qui gère la génération de la grille du jeu (tableau à 2 dimensions).

-le module Saisie qui gère le tableau de stockage des coordonnées durant le jeu .

-le module main qui contient la fonction principal du jeu .

Implémentation :

Nous allons voir dans cette partie les principales fonctions du projet et expliquer nos choix sur certains points et donner quelque exemple de fonctions utilisées :

Dico.c :

Pour cette partie nous avons repris nos fonctions déjà implémenté durant le DM d'algorithme des arbres sur les arbres lexicographique que nous avons eu il y a quelques semaines.

```
int rechercher(Tarbre tr, char mot[]){
    if(tr == NULL){
        return 0;
    }
    if (mot[0] == tr->lettre){
        if(mot[0]=='\0'){
            return 1;
        }
        else{
            return rechercher(tr->fils,mot+1);
        }
    }
    else{
        if(mot[0] < tr->lettre){
            return rechercher(tr->fg, mot);
        }
        else{
            return rechercher(tr->fd, mot);
        }
    }
}
```

Exemple ici , la fonction recherche() qui permet de savoir si un mot est présent dans un arbre ou non , nos fonctions de ce module utilisent beaucoup de récursivité .

Grille.c

Au niveau de la grille nous avons décidé de la représenter par un tableau à 2 dimensions avec des lettres en minuscule car le dictionnaire fourni était aussi composé de mot en minuscule. Pour générer les lettres aléatoirement avec les probabilités qu'on nous a fourni , nous avons fait des tableaux contenant des lettres rangés par probabilité :

```
char lettre_aleatoire(void){
    int cas;
    int i;
    int alea;

    char tab0[]={'e'};
    char tab1[]={'t'};
    char tab2[]={'a','i','n','o','s'};
    char tab3[]={'r'};
    char tab4[]={'h'};
    char tab5[]={'d', 'l'};
    char tab6[]={'c', 'm', 'u'};
    char tab7[]={'b', 'f', 'g', 'p', 'w', 'y'};
    char tab8[]={'j', 'k', 'q', 'v', 'x', 'z'};
```

Puis en tirant un nombre aléatoirement entre 0 et 100 , nous avons mis des cas correspondant au probabilité de l'énoncé dans des conditions de cette manière :

```
alea= rand()%100;

if ( alea >= 0 && alea <= 10 ){
    cas = 0;
}

if ( alea >= 11 && alea <= 18){
    cas = 1;
}

if ( alea >= 19 && alea <= 53){
    cas = 2;
}

if ( alea >= 54 && alea <= 58){
    cas = 3;
}

if ( alea >= 59 && alea <= 66){
    cas = 4;
}
```

Puis suivant le cas ou l'on tombait , on tire un chiffre dans le tableau correspondant à ce cas :

```
switch(cas){
    case(0) : return tab0[0];
              break;

    case(1) : return tab1[0];
              break;

    case(2) : i=rand()%5;
              return tab2[i];
              break;

    case(3) : return tab3[0];
              break;

    case(4) : return tab4[0];
              break;

    case(5) : i=rand()%2;
              return tab5[i];
              break;
```

Tout cela est contenu dans la fonction lettre_aleatoire() .

Regle.c :

Au niveau des fonctions qui gère les règles du jeu , il y en a 2 , l'une s'appelant coup_legal() qui vérifie si un coup n+1 est légal par rapport à un coup n , en vérifiant toute les coordonnés possible pour n+1 lorsque l'on part de n . Cette fonction vérifie aussi si une lettre a été utilisé plusieurs fois en consultant un tableau tab_x et tab_y qui contiennent les coordonnées des coups joués par l'utilisateur.

L'autre fonction est verif_mot , elle permet de vérifier si un mot est valide par rapport à un indice qui est déclaré dans la boucle principal du jeu et qui représente le résultat de la fonction coup_legal(). Elle vérifie si le mot passé en paramètre est bien dans le dictionnaire qu'on nous a fourni et elle vérifie aussi si le mot n'a pas déjà été trouvé par le joueur précédemment dans la partie. Cette fonction augmente aussi le score qu'on lui passe en paramètre en fonction de la taille du mot trouvé .

```c
int verif_mot(int taille, int *indice, Tarbre dico, Tarbre *mot_trouve, char *mot, int *score ,int tab_x[], int tab_y[]){
    int t;
    assert(taille == 4);
    if(*indice == 1 && rechercher(dico, mot) == 1 && rechercher(*mot_trouve, mot) == 0){
        /* 1 alors on ajoute le mot à une liste des mots trouvé puis on affiche que c'est trouvé + augmentation du score*/
        t = strlen(mot);
        *score = *score + pow(2, (t-3));
        /*ajouter mot dans l'arbre mot_trouve */
        ajout_arbre(mot_trouve, mot, 0);

        init_tab_coord(taille, tab_x, tab_y);
        *indice = 2;
        return 1;
    }
    else{
        init_tab_coord(taille, tab_x, tab_y);
        *indice = 2;
        return 0;
    }
}
```

Graphique.c :

Au niveau de l'affichage nous avons utilisé la librairie ncurses avec un affichage assez basique avec la grille et les informations sur les tours restants , le mot que l'on est entrain de construire et le nombre de point que l'on a durant la partie .

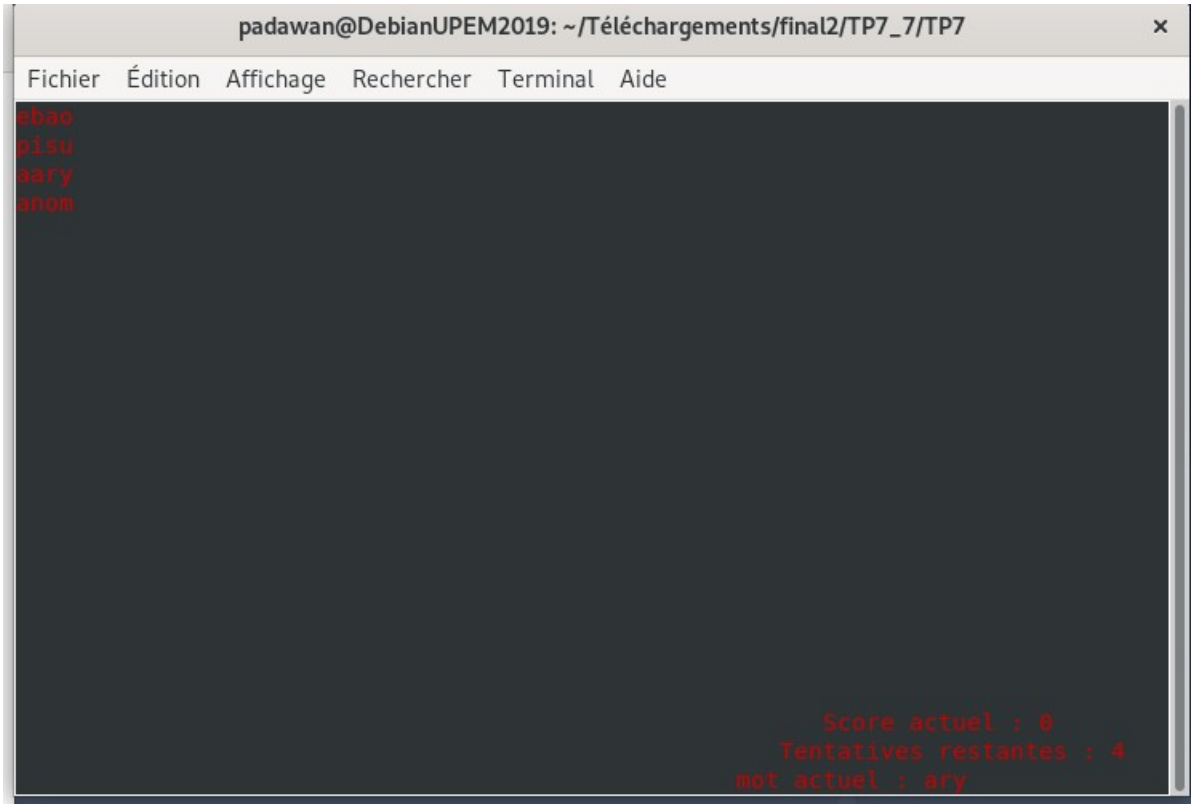
```c
void message_score_tour(int points, int tour, char mot[]){
    mvprintw(LINES - 3, COLS - 24, "Score actuel : %d", points);
    mvprintw(LINES - 2, COLS - 27, "Tentatives restantes : %d", tour);
    mvprintw(LINES - 1, COLS - 30, "mot actuel : ", "%s", mot);
    refresh();
    sleep(1);
}
```

Main.c :

Au niveau du main , nous avons une boucle principal qui s'arrête lorsque le nombre de tour est écoulé. Dans cette boucle nous faisons appelle à la partie graphique afin d'afficher la grille , nous récupérons les coordonnées des clics de l'utilisateur afin de composé le mot que l'utilisateur souhaite faire. Lorsque l'utilisateur soumet le mot , alors nous appelons les fonctions qui vérifie si le mot est valide ou non à l'aide des fonctions du modules de Regle.c

Guide utilisateur :

Lors du lancement du programme , une taille vous sera demandé , il faut saisir 4 ici car c'est la seule taille que propose notre programme.
Lorsque la fenêtre du jeu s'ouvre , il vous suffira de cliquer à l'aide d'un clic droit sur les lettres que vous voulez sélectionner , lorsque vous cliquez sur une lettre alors elle apparaît en bas a droite dans mot actuel .
Lorsque vous avez sélectionné entièrement le mot trouvé , alors pour le soumettre il vous suffit d'appuyer sur la touche 'entrer' , un message apparaîtra afin de voir si le mot est valide ou non .
Pour quitter le jeu il vous suffit d'appuyer sur la touche 'q'.



Conclusion :

Pour conclure ce TP nous aura permis d'approfondir les notions de modularisation et autres notions comme les assertions ou autre que nous avons vu en cour durant ces dernières semaines . Le fait que le TP soit moins guidé que d'habitude a eu ces avantages et ces inconvénients . Au niveau des difficultés rencontrés , nous avons énormément de bug à corriger sur nos fonctions dans règle.c , en effet le fait d'avoir fait la partie graphique à la fin du projet a fait que nous n'avons pas pu tester les fonctions avant de faire celle-ci.

De plus le fait d'avoir eu un DM en algorithme des arbres sur les arbres lexicographique nous a vraiment aidé dans le développement de ce projet . Malheureusement par manque de temps nous n'avons pas eu le temps d'améliorer la partie graphique et de nous pencher sur les améliorations qu'étaient proposées par le sujet.

Annexe 1 :

Pour compiler le programme il suffit d'aller dans le dossier TP7 contenant les fichier.c et .h , puis en ouvrant le terminal dans ce dossier on tape la commande « make » .

./Boggle -t
Cette option permet de lancer les fonctions de test de notre code.

Annexe 2 :

Tout nos programmes se compilent avec les commandes en annexe 1 , puis on les exécute avec la commande ./Boggle