Rapport

Programmation Impérative 2

Projet 2019-2020 :
Scrabble

Nous avons décidé ensemble de prendre le Scrabble comme projet. C'est un jeu connu par beaucoup de personnes, cela nous intéressait et nous voulions rester dans un concept de réalisation de jeu.

1 Plateau

Premièrement, nous avons regardé les règles pour avoir une idée de l'organisation de notre plateau. En effet, chaque cases de notre plateau possèdent une valeur différente. De plus chaque lettres possèdent un score différent et une occurence différente, c'est à dire que plus la lettre peut être utilisé dans un mot, plus il y'a de chances de la retrouver dans notre main. Par ailleurs, il a fallut penser au joker, qui peut prendre comme valeur n'importe quelle lettre.

L'initialisation du plateau :

```
    set_type(9,9,2,plateau);
set_type(13,9,2,plateau);
set_type(5,13,2,plateau);
set_type(9,13,2,plate

////////////////////////////////////////////////////////////////////////////////////////////////////

    ajout_lettre_plateau(0,0,'S',plateau);
ajout_lettre_plateau(0,1,'I',plateau);
ajout_lettre_plateau(0,2,'F',plateau);
ajout_lettre_plateau(0,4,'L',plateau);
ajout_lettre_plateau(0,5,'E',plateau);

////////////////////////////////////////////////////////////////////////////////////////////////////
```

Premièrement, on définit chaque coordonnées de chaque cases de notre plateau. Ensuite on lui affecte une valeur qui est, soit mot compte double/triple ou lettre compte double/triple. Puis on a juste a placé chaque lettres à l'endroit souhaiter, avec la valeur de la lettre une fois calculée.L'initilisation de notre plateau se fait dans le main.

2 Structures

Ici notre fichier prototype contenant l'ensemble de nos strctures et fonctions ;

```
//////////////////////////////////////////STRUCTURES////////////////////////////////////////////////

////MOT//////////////////////////////


typedef struct mot_t mot_t;
struct mot_t{
char lettres[16];
int n;
};


////NODE//////////////////////////


typedef struct node_t node_t;
struct node_t{
mot_t * mot;
node_t * suivant;
};
```

```
typedef struct node_t * dico_t; //ou pointeur liste

////LISTE_OCCU//////////////////////

typedef struct node_occu_t node_occu_t;
struct node_occu_t{
mot_t * mot;
int position;
int sens;//1 : horizontal / 0 : vertical
node_occu_t * suivant;
};

typedef struct node_occu_t * dico_occu_t;


////CASE//////////////////////////////


typedef struct case_t case_t;
struct case_t{
//coordonnée
int iPlateau;

//type de case (double, triple) : normal -> 0 / lettre : double -> 1 / triple -> 2 / mot : double -> 3 / t
int type;
char lettre;
int score;
};

typedef struct case_mot_t case_mot_t;
struct case_mot_t{
case_t lettres[16];
int n;
};



////PLATEAU//////////////////////////

typedef struct plateau_t plateau_t;
struct plateau_t{

case_t plateau[225];
plateau_t * suivant;
};

typedef struct plateau_t * liste_plateau_t;

////COUP//////////////////////////////


typedef struct mot_coup_t mot_coup_t;
struct mot_coup_t{

case_mot_t * mot;
mot_coup_t * suivant;
int scoreMot;
};
typedef struct mot_coup_t * liste_mot_coup_t;

typedef struct coup_t coup_t;
struct coup_t{
```

```
int scoreCoup;
liste_mot_coup_t mots;
coup_t * suivant;
int scrabble;//0 : oui / 1 : non
};

typedef struct coup_t * liste_coup_t;



////////////////////////////////////////////////PROTOTYPES/////////////////////////////////////////

dico_t init_dico();
int empty_dico(dico_t);
mot_t * creer_mot(char *);
dico_t insertion_dico(dico_t, char *);
void print_dico(dico_t);
case_t * getCase(int, int, case_t *);
void ajout_lettre_plateau(int, int, char, case_t *);
void set_type(int, int, int, case_t *);
void valeur_lettre(int, int, case_t *);
void valeur_lettre_plateau(case_t *);
void valeur_lettre_main(case_t *);
void init_plateau(case_t *);
void print_plateau(case_t *);
void affiche_plateau_entier(case_t *, case_t *);
void init_hand(case_t *);
void ajout_lettre_hand(int, char, case_t *);
int lettre_dans_main(case_t *, char);
void print_hand(case_t *);
int nombre_lettre(char *);
int est_dans(char *, dico_t);
int mot_egal(char *, char *);
int appartient(char *, dico_t);
int contient(char *, char *);
int existe(char *);
char * recuperation_verticale(case_t *, int, char *);
char * recuperation_horizontale(case_t *, int, char *);
int recuperation_lettre_up(case_t *, int);
int recuperation_lettre_left(case_t *, int);
int occurence_lettre_mot(char *, char);
dico_t groupe_mot_plateau(case_t *, dico_t);
void algorithme_coup(dico_t, case_t *, case_t *, liste_coup_t, case_t *, case_t *);
int position_lettre_mot(char, char *, int);
liste_plateau_t insertion_liste_plateau(liste_plateau_t, case_t *);
int empty_liste_plateau(liste_plateau_t);
liste_plateau_t init_liste_plateau();
void affiche_plateau(case_t *);
void affiche_liste_plateau(liste_plateau_t);
void copy_plateau(case_t *, case_t *);
void copy_hand(case_t *, case_t *);
void affiche_hand(case_t *);
int effacer_lettre_hand(case_t *, char);
int effacer_joker_hand(case_t *);
liste_coup_t _liste_coup();
int main_pleine(case_t *);
case_mot_t * creer_case_mot(char *, int, int, case_t *);
liste_mot_coup_t insertion_liste_mot_coup(liste_mot_coup_t, mot_coup_t *);
liste_mot_coup_t init_liste_mot_coup();
void print_liste_mot_coup(liste_mot_coup_t);
int return_valeur_lettre(char);
int main_vide(case_t *);
```

```
liste_coup_t insertion_liste_coup(liste_coup_t, coup_t *);

int empty_liste_coup(liste_coup_t);
int empty_liste_mot_coup(liste_mot_coup_t);
int trou_main(case_t *);
```

3 Les parcours

Voici l'algorithme des différents modes de parcours sur le plateau, de manière horizontale ou verticale et en fonction des différents groupes de mots qui sont disponibles sur le plateau. Il cherche à effectué le meilleur coup possible.

```
if(choixParcours!=0){////////////////////////PARCOURS MONTANT/////////////////////////////////////////////

while(iParcoursMot){////////////////////TRI MONTANT/////////////////////////////////////////////////////

iParcoursMot--;
iPlateauCopy-=15;

//printf("(%c,%d) : %d\n", dicoCopy->mot->lettres[iParcoursMot], iParcoursMot, iPlateauCopy);

if(iPlateauCopy<0){
exit=1;
break;
}

if((plateau[iPlateauCopy].lettre!=dicoCopy->mot->lettres[iParcoursMot]) && plateau[iPlateauCopy].lettre!='
exit=1;
break;
}

if(plateau[iPlateauCopy].lettre==' '){

if(!copyYetHand){
copy_hand(hand,handCopy);
copyYetHand=1;
}

if(effacer_lettre_hand(handCopy, dicoCopy->mot->lettres[iParcoursMot])==0){
if((effacer_joker_hand(handCopy))==0){
exit=1;
break;
}
}
}
}////////////////////////////////////////TRI MONTANT/////////////////////////////////////////////////////

}////////////////////////////////////////PARCOURS MONTANT/////////////////////////////////////////////

iParcoursMot=iMot;
iPlateauCopy=iPlateau;

if(exit==0){
//printf("position : %d MOT(%d) : %s rentre hauteur\n", i, iMot ,dicoCopy->mot->lettres);
}

if(choixParcours!=1 && exit==0){////////////PARCOURS DESCENDANT/////////////////////////////////////////////

while(dicoCopy->mot->lettres[iParcoursMot+1]!=' ' && dicoCopy->mot->lettres[iParcoursMot+1]!='\0'){///////

iParcoursMot++;
iPlateauCopy+=15;
```

```
if(iPlateauCopy/15>14){
exit=1;
break;
}

if((plateau[iPlateauCopy].lettre!=dicoCopy->mot->lettres[iParcoursMot]) && plateau[iPlateauCopy].lettre!='
exit=1;
break;
}

if(plateau[iPlateauCopy].lettre==' '){

if(!copyYetHand){
copy_hand(hand,handCopy);
copyYetHand=1;
}

if(effacer_lettre_hand(handCopy, dicoCopy->mot->lettres[iParcoursMot])==0){
if((effacer_joker_hand(handCopy))==0){
exit=1;
break;
}
}
}

}////////////////////////////////////////////////////////////////////////////////////////////////////////////


}/////////////////////////////////////////PARCOURS DESCENDANT/////////////////////////////////////////

if(!copyYetHand)
exit=1;

if(exit==0){
//printf("position : %d MOT(%d) : %s rentre haut bas\n", i, iMot ,dicoCopy->mot->lettres);
//affiche_hand(handCopy);
}

if(exit==0){
iParcoursMot=iMot;
iPlateauCopy=iPlateau;
copy_hand(hand,handCopy);
copy_plateau(plateau,plateauCopy);
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////

if(choixParcours!=0 && exit==0){//////////////////////PLACE MONTANT

while(iParcoursMot){

iParcoursMot--;
iPlateauCopy-=15;

if(plateauCopy[iPlateauCopy].lettre==' '){

if(effacer_lettre_hand(handCopy, dicoCopy->mot->lettres[iParcoursMot])==1){
plateauCopy[iPlateauCopy].lettre=dicoCopy->mot->lettres[iParcoursMot];
}
else if(effacer_joker_hand(handCopy)==0){
exit=1;
break;
}
```

```
else
plateauCopy[iPlateauCopy].lettre='?';


if(!iParcoursMot){

if(iPlateauCopy>14){

if(plateauCopy[iPlateauCopy-15].lettre!=' '){
recuperation_verticale(plateauCopy,iPlateauCopy,motExiste);

if(!existe(motExiste)){

if(effacer_joker_hand(handCopy)==1){

handCopy[trou_main(handCopy)].lettre=plateauCopy[iPlateauCopy].lettre;
plateauCopy[iPlateauCopy].lettre='?';
recuperation_verticale(plateauCopy,iPlateauCopy,motExiste);

if(!existe(motExiste)){
exit=1;
break;
}
}
else{
exit=1;
break;
}
}
}
}
}

if( (iPlateauCopy%15!=0 && plateauCopy[iPlateauCopy-1].lettre!=' ') || (iPlateauCopy%15!=14 && plateauCopy

recuperation_horizontale(plateauCopy,iPlateauCopy,motExiste);

if(!existe(motExiste)){

if(effacer_joker_hand(handCopy)==1){

handCopy[trou_main(handCopy)].lettre=plateauCopy[iPlateauCopy].lettre;
plateauCopy[iPlateauCopy].lettre='?';
recuperation_horizontale(plateauCopy,iPlateauCopy,motExiste);

if(!existe(motExiste)){
exit=1;
break;
}
}
else{
exit=1;
break;
}
}
}
}
}
```

Ce sont nos 2 modes de parcours decendant et montant. Il regarde en fonction de notre plateau et des lettres qui sont positionées sur le plateau formant un groupe de mots, si il peut placé un mot qui est contenu dans le dictionnaire.

De plus il verifie aussi si le mot respecte bien les limites du plateau. Cepenadant pour chacun des parcours nous étions censé stocker le coup, mais nous n'avons pas pu y arriver faute de temps.
De plus nous nous sommes rendu compte à la toute fin des limites de la fonction récupérationleft qui ne parcours pas toute les cases de la ligne et donc le résultat des coups possibles peut être faussé.
Enfin, nous comptions implémenter à la fin, c'est pour cela qu'ils sont absents de notre programme.

4 Main

Voici un extrait de notre main :

```
Voici un extrait de notre main  : dico_t dico = init_dico();

FILE *dicoTxt;
dicoTxt = fopen("dic.txt","r");
char mot[16];

while(fscanf(dicoTxt, "%s", mot) != EOF){
listeTmp=liste;

while(listeTmp){
if(contient(mot,listeTmp->mot->lettres)){
dico=insertion_dico(dico, mot);
break;
}
listeTmp=listeTmp->suivant;
}
}
```

Ici il copie les mots contenus dans notre fichier dico.txt en fonction du tri des groupes de lettres préentes sur notre plateau.

5 Makefile

```
CC = gcc
CFLAGS = -Wall
EXEC=scrabble

all: $(EXEC)

scrabble: fonctions.o main.o
  $(CC) -o scrabble fonctions.o main.o

main.o: main.c scrabble.h
    $(CC) -o main.o -c main.c $(CFLAGS)


fonctions.o:  fonctions.c scrabble.h
        $(CC) -o fonctions.o -c fonctions.c $(CFLAGS)


clear:
    rm *.o *~ core
```

6 Conclusion

Ce projet nous a permis de renforcer notre niveau en C. Malgré cela nous avons rencontré pas mal de difficultés, notamment dans les allocations de mémoires et l'utilisation de pointeurs. Par exemple lorsqu'il fallait modifier des variables sans passer par le main.Ce programme très fourni en code nous a permis de prendre conscience des différents

problèmes d'organisations et d'inter-connexions des fonctions. Malheuresment notre projet n'a pas pu aboutir au résultat souhaité et cela est d'autant plus dommage et frustrant car nous savons que nous étions proches du but. Néanmoins nous sommes fiers du programme que nous avons produits, dépassant de loin ce que nous pensions être capable de réaliser.