Mezouar Mikael Université Gustave Eiffel Peinzi Ferdyl

Rapport TP5 : Perfectionnement en programmation C

23/03/2021 L2 informatique

Mezouar Mikael Université Gustave Eiffel Peinzi Ferdyl

Introduction :

L'objectif de ce TP était de réaliser le jeu du serpent, autrement appelé le Snake. Le principe du Snake est simple. Un serpent se balade de case en case dans une zone délimitée tant qu'il ne sort pas du cadre. Le serpent peut grossir en taille en mangeant des pommes qui apparaissent de façon aléatoire dans la zone délimitée. Dans ce TP nous avons retravaillé avec la librairie *Ncurses* avec laquelle on a interagit en temps réel (avec le déplacement du serpent commandé par l'utilisateur), nous avons également utilisé la notion de *liste chaînée* et nous avons mis en place une compilation paramétrée avec un *Makefile*. Nous allons voir dans ce rapport comment nous avons conçu ce projet.

Exercice 1:

2)Le découpage de départ :

-le module case avec le type Case .
-le module direction avec le type Direction
-le module serpent avec le type Serpent
-le module pomme avec le type Pomme
-le module monde avec le type Monde
-le module graphisme qui s'occupe de l'affichage du jeu .

3) historique des changements :

-6 modules au départ

-7 modules à la fin car au fil du projet , on s'est rendu compte qu'il fallait un module qui dirige les paramètres du jeu (taille du plateau , nombre de pomme , durée d'un tour , taille du serpent au départ) que l'on a nommé menu car il dirige les paramètres que l'on pourrait mettre au choix de l'utilisateur sous forme d'un menu .

Exercice 2 (Déclaration des types):

Dans cette partie on a défini les bases de notre projet, c'est-à-dire les types que l'on allait utiliser pour nos programmes. On a donc mis en place plusieurs structures. Seul le type Direction est de type énuméré. Les structures de notre programme sont de la forme suivante :

```
typedef struct{
    int x;
    int y;
}Case;
```

Figure 1 : Structure du type Monde

```
typedef struct {
    int nb_ligne;
    int nb_colonne;
    Serpent serpent;
    EnsembleP pomme_presente; /*ensemble des pommes présentes*/
    int taille_serpent;
    int nb_pomme_mangee;
} Monde;
```

Figure 2 : Structure du type Monde

```
typedef struct {
    Case pos;
} Pomme;

typedef struct pommes {
    Pomme pomme;
    struct pommes *suivant;
} Pommes, *EnsembleP;
```

Figure 3 : Structures du type Pomme

Dans ces structures on voit bien que chaque variable est définie avec son type au préalable.

Pour le type Direction la définition du type est différent, les variables n'ont pas vraiment de type, et elles sont directement utilisées comme valeur au sein des programmes. On peut le remarquer ci-dessous :

```
typedef enum {
    NORD,
    SUD,
    EST,
    OUEST
} Direction;
```

Figure 4 : Structure du type Direction

```c
Case prochaine_case(Serpent serpent) {
    Case retour;

    retour.x = serpent->coord.x;
    retour.y = serpent->coord.y;
    switch (serpent->sens) {
        case NORD : retour.y--; break;
        case SUD : retour.y++; break;
        case EST : retour.x++; break;
        case OUEST : retour.x--; break;
        default : break;
    }
    return retour;
}
```

Figure 5 : Exemple d'utilisation du type Enum

On constate bien que les données rentrées dans la structure du type Direction sont bien utilisées comme des valeurs.

Étant donné qu'il est demandé dans l'énoncé d'utiliser un *Makefile* nous avons dû décomposer le programme en plusieurs modules (décomposition qui a évolué au fil du projet) : on a *Case*, *Menu*, *Monde*, *Pomme*, *Serpent* et *Graphique*. Cette décomposition a donné naissance à plusieurs fichiers. On a des fichiers « *.c », des « *.h » et des fichiers « *.o ». Par exemple pour le module *Menu* on aura les fichiers Menu.c, Menu.h et Menu.o.

Exercice 3 (Mécanique du jeu):

1)

```c
Pomme pomme_gen_alea(int n, int m) {
    int abs;
    int ord;
    Pomme new;

    abs = rand() % n;
    ord = rand() % m;
    new.pos.x = abs;
    new.pos.y = ord;
    return new;

}
```

Pour cette fonction nous avons utilisé la fonction rand afin de générer une pomme aléatoirement par rapport au coordonnée x et y sur le quadrillage.

2)

```c
int ajouter_pomme_monde(Monde *mon) {
    Pomme nouvelle;
    EnsembleP temp;

    nouvelle = pomme_gen_alea(mon->nb_ligne, mon->nb_colonne);
    if (!case_libre(*mon, nouvelle.pos))
        return 0;
    if (!(temp = (EnsembleP)malloc(sizeof(Pommes))))
        exit(EXIT_FAILURE);
    temp->pomme.pos.x = nouvelle.pos.x;
    temp->pomme.pos.y = nouvelle.pos.y;
    temp->suivant = mon->pomme_presente;
    mon->pomme_presente = temp;
    return 1;
}
```

Pour cette fonction on utilise la fonction pomme_gen_alea que l'on a implanté juste avant , nous avons aussi créer une fonction case_libre qui vérifie si la case du monde pris en paramètre est bien libre , si elle n'est pas libre alors la fonction renvoi 0 , sinon elle renvoi 1 et ajoute la pomme au monde.

3)

```c
Serpent init_serpent(Monde mon) {
    Serpent retour;
    Serpent temp;
    int i;

    temp = NULL;
    for (i = 0 ; i < mon.taille_serpent ; i++) {
        if (!(retour = (Serpent)malloc(sizeof(Morceau))))
            exit(EXIT_FAILURE);
        retour->coord.x = mon.nb_colonne / 2 + i - mon.taille_serpent - 1;
        retour->coord.y = mon.nb_ligne / 2;
        retour->sens = EST;
        retour->corps = temp;
        temp = retour;
    }
    return retour;
}
```

Pour cette fonction nous avons simplement initaliser le serpent au milieu du quadrillage du jeu avec comme direction initiale l'Est comme il est demandé dans la consigne.

4)

```
Monde init_monde(Option opt) {
    Monde retour;
    int i;

    retour.pomme_presente = NULL;
    retour.nb_ligne = opt.nb_ligne;
    retour.nb_colonne = opt.nb_colonne;
    retour.taille_serpent = opt.taille_serpent;
    retour.serpent = init_serpent(retour);


    for (i = 0 ; i < opt.nb_pommes ; i++) {
        while (!ajouter_pomme_monde(&retour));
    }
    retour.nb_pomme_mangee = 0;
    return retour;
}
```

Pour cette fonction nous avons modifier le paramètre de la fonction en mettant une un valeur de type option (structure que nous avons implanté avec les paramètres du jeu) , afin d'initialiser un monde par rapport à des options qu nous avons initialisé et afin de pouvoir les changer à notre guise .

5)

```
int deplacer_serpent(Monde *mon) {
    Serpent temp_s;
    Case nouvelle;
    Case temp_c;

    nouvelle = prochaine_case(mon->serpent);
    if (!case_libre(*mon, nouvelle))
        return 0;
    for (temp_s = mon->serpent ; mon->serpent ; mon->serpent = mon->serpent->corps) {
        temp_c = mon->serpent->coord;
        mon->serpent->coord = nouvelle;
        nouvelle = temp_c;
    }
    mon->serpent = temp_s;
    return 1;
}
```

Pour cette fonction , nous avons créé une autre fonction qui est prochaine_case , on regarde avec case_libre si la case suivante est libre on fait le déplacement du serpent .

6) Voir le code pour la fonction. Pour cette fonction nous prenons un monde en paramètre elle modifie le serpent du monde mon de sorte à le faire avancer suivant sa direction vers une case occupée par une pomme. Elle renvoie 1 si la tête du serpent arrive bien sur une case occupée par une pomme du quadrillage. Elle supprime la pomme ainsi mangée du monde mon grâce au free à la fin.

7)

```
int mort_serpent(Monde mon) {
    Serpent temp_s;
    Case nouvelle;

    nouvelle = prochaine_case(mon.serpent);
    if (nouvelle.x >= mon.nb_colonne || nouvelle.x < 0)
        return 1;
    if (nouvelle.y >= mon.nb_ligne || nouvelle.y < 0)
        return 1;
    for (temp_s = mon.serpent->corps ; temp_s ; temp_s = temp_s->corps) {
        if (compare_case(nouvelle, prochaine_case(temp_s)))
            return 1;
    }
    return 0;
}
```

Pour cette fonction on vérifie si le serpent vérifie une condition de mort , c'est à dire une sortie du quadrillage ou si la case est occupé par son corps .On renvoie 1 si il meurt , sinon on renvoie 0 .

Exercice 4 (Graphisme) :

1)

```
void afficher_quadrillage(Monde mon) {
    int i;
    int j;

    for (i = 0 ; i < mon.nb_ligne ; i++) {
        for (j = 0 ; j < mon.nb_colonne ; j++)
            affiche_carre((COTE - 1) * i, (COTE - 1) * j, COTE);
    }
}
```

Pour cette fonction nous avons utilisé la librairie Ncurses afin de faire l'affichage avec les caractère +,-,| et des espaces comme nous l'avions fais dans les précédents TP pour représenter le quadrillage.

2)

```
void afficher_pomme(Pomme pom) {
    int centre;

    centre = COTE / 2;
    attron(COLOR_PAIR(1));
    mvaddch(pom.pos.y * (COTE - 1) + centre,
            pom.pos.x * (COTE - 1) + centre, 'o');
    attroff(COLOR_PAIR(1));
}
```

Pour cette fonction nous avons décidé de représenter les pommes avec le caractère 'o' avec une couleur unique (le rouge dans notre programme).

3)

```
void afficher_serpent(Serpent ser) {
    int centre;

    centre = COTE / 2;
    attron(COLOR_PAIR(TETE));
    mvaddch(ser->coord.y * (COTE - 1) + centre,
            ser->coord.x * (COTE - 1) + centre, 'o');
    ser = ser->corps;
    attroff(COLOR_PAIR(TETE));
    attron(COLOR_PAIR(MORCEAU));
    while (ser) {
        mvaddch(ser->coord.y * (COTE - 1) + centre,
                ser->coord.x * (COTE - 1) + centre, 'o');
        ser = ser->corps;
    }
    attroff(COLOR_PAIR(MORCEAU));
}
```

Pour cette fonction nous avons décidé de représenter le serpent avec le caractère ‘o’ , avec la tête d’une couleur (vert) et le corps avec aussi le caractère ‘o’ mais d’une autre couleur (le cyan).

4)

```
void afficher_monde(Monde mon) {
    EnsembleP temp;

    afficher_quadrillage(mon);
    for (temp = mon.pomme_presente ; temp; temp = temp->suivant)
        afficher_pomme(temp->pomme);
    afficher_serpent(mon.serpent);
    mvprintw((COTE - 1) * mon.nb_ligne + 1, COLS / 2 - 10,
             "Le nombre de pomme mangee est : %2d", mon.nb_pomme_mangee);
}
```

Pour cette fonction , on utilise tout simplement toute les fonctions que l’on a implémenté avant avec le monde que l’on veut représenter en paramètre, en rajoutant le nombre de pomme mangée .

Exercice 5 (Assemblage du jeu) :

```
void jeu(void){
    Option a=init_option();
    int touche;
    Monde partie = init_monde(a);
    clear();
    noecho();

    afficher_monde(partie);
    refresh();
    getch();
    cbreak();
    nodelay(stdscr, TRUE);
    while(1){
        if (mort_serpent(partie)) {
            clear();
            mvprintw(LINES / 2, COLS / 2 - 4, "PERDU !");
            refresh();
            cbreak();
            nodelay(stdscr, FALSE);
            getch();
            break;
        }
        if (!deplacer_serpent(&partie)) {
            if (manger_pomme_serpent(&partie)==0)
                while (!ajouter_pomme_monde(&partie));
        }
        afficher_monde(partie);
        refresh();
        touche = getch();
        switch (touche) {
            case 'q' : partie.serpent->sens = OUEST; break;
            case 'd' : partie.serpent->sens = EST; break;
            case 's' : partie.serpent->sens = SUD; break;
            case 'z' : partie.serpent->sens = NORD; break;
            case ' ' : while (getch() != ' '); break;
            default : break;
        }
        usleep(a.duree_tour);
    }
}
```

Pour l'assemblage du jeu , nous avons créé une fonction jeu dans laquelle nous avons suivi le pseudo code donné par la consigne du TP .

Conclusion :

Pour conclure ce TP aura été le plus compliqué depuis le début , il nous aura permis de travailler en faisant de la programmation modulaire , en effet c'est la première que nous avons du faire un telle projet avec différents fichiers , cela a été compliqué au début à prendre en main , mais cela aura était bénéfique pour nous afin de bien comprendre et mettre en pratique cette notion du cour. Enfin ce TP nous a aussi permis de voir la représentation creuse qui s'avère vraiment pratique pour représenter les objets qui peuvent être lourd en mémoire (un quadrillage très grand par exemple) , jusqu'ici nous étions habitué à représenter les quadrillages par des tableaux à double dimensions .

Durant ce projet nous avons rencontré énormément de difficulté , tout d'abord au niveau du temps , nous n'avons pas vraiment eu le temps d'aborder les améliorations de l'exercice 6.Ensuite nous avons rencontré beaucoup de problème pour bien inclure tout

les fichiers entre eux. Enfin nous avons aussi beaucoup du modifier les structures afin de stocker tout ce qui était nécessaire à la conception du Snake. De plus il reste un bug dont nous n'avons pas réussi a résoudre , c'est le fait que lorsque l'on va dans la direction opposé avec le Snake , le programme bug , nous n'avons pas eu le temps de se pencher énormément sur ce problème .

Annexe 1 :

Pour compiler le programme il suffit d'aller dans le dossier projet_tp5 contenant le dossier include et src, puis en ouvrant le terminal dans ce dossier on tappe la commande « make » .

Pour modifier les paramètres du jeu il suffit de changer les valeurs dans la fonction init_option dans le fichier menu.c (on peut modifier la taille de base du serpent , le nombre de colonne et de ligne du quadrillage ou le nombre de pomme par exemple) .

Annexe 2 :

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