Mezouar Mikael Université Gustave Eiffel Dos Santos Loic

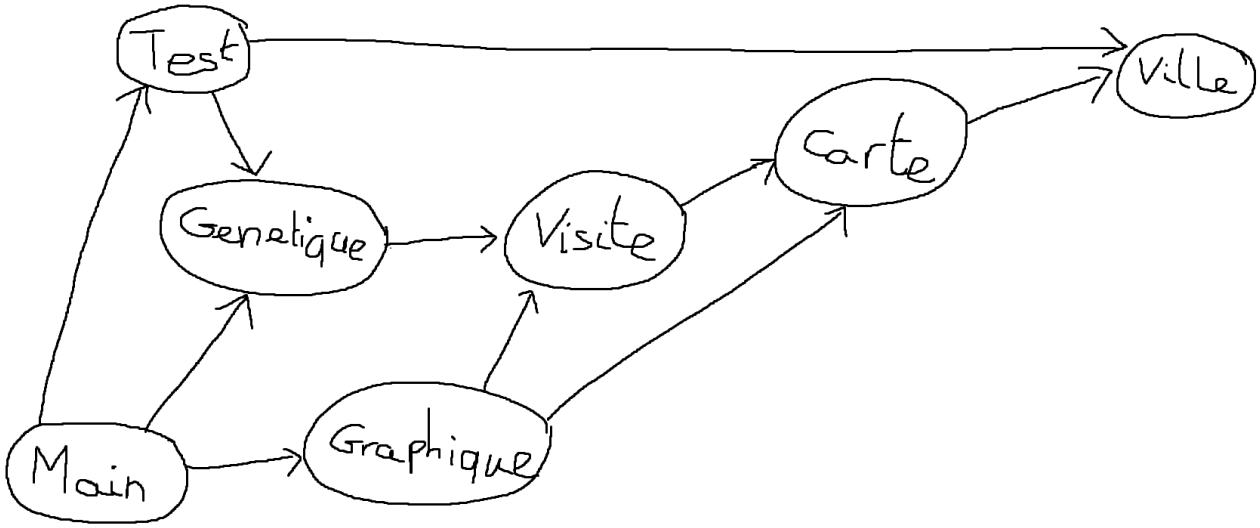
Rapport TP9 : Perfectionnement en programmation C

03/05/2021 L2 informatique

Introduction :

L'objectif de ce TP est de résoudre le problème du voyageur de commerce ,le problème du voyageur de commerce décrit une situation qui prend en entrée un ensemble de villes avec leur position géographique, et qui consiste à fournir en réponse un ordre de visite de ces villes qui soit le plus court possible. Nous allons donc voir dans ce rapport comment nous avons réussi à implémenter un programme qui résout le problème de départ .

Modularisation :



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

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

-le module test qui comporte les tests .

-le module Graphique qui gère la partie graphique du problème avec la librairie MLV .

-le module Génétique qui comporte la partie génération par génétique du projet

-le module Visite qui comporte la structure Visite et des fonctions qui manipulent les visites.

-le module Carte qui contient la structure carte et qui gère notamment le fichier d'entré fourni par l'utilisateur.

-le module Ville qui comporte la structure Ville

Implémentation :

1)Les principales structures et autre spécification

Au niveau des structures que nous avons utilisé pour implanter le programme , nous avons suivi les structures que le sujet nous suggéré :

```
typedef struct{
    int taille;
    Ville *tab;
}Carte;
```

```
typedef struct{
    int taille;
    Ville *tab;
}Visite;
```

```
typedef struct{
    char nom_ville[128];
    int x;
    int y;
    int etiquette;
}Ville;
```

Par ailleurs nous avons ajouté une structure pour la programmation génétique , afin de gérer les 3 paramètres alpha , bêta et gamma :

```
typedef struct {
    int alpha;
    int beta;
    int gamma;
} Parametre;
```

Au niveau du fonctionnement du programme , le programme prend un fichier txt en entré de cette forme (avec une ville par ligne avec le nom de la ville et les coordonnées :

```
Solitude 1 1
Epervine 2 2
Markarth 3 1
Morthal 5 1
Faillaise 1 2
Aubétoile 7 1
Vendeaume 0 0
Fordhiver 1 8
Rivebois 4 6
Helgen 6 4
Fort-Ivar 5 5
Rorikbourg 1 4
Pondragon 7 7
Pierre-de-Shor 9 9
Kyne 8 1
Corberoc 4 4
```

Cet fonction permet de lire le fichier.txt et de récolter les informations nécessaire dans la structure principal du programme (dans la structure carte) .

```
Carte init_carte(char* nom_fichier) {
    Carte c;
    FILE *f;
    int taille, i;
    char lettre;

    taille = 0;
    i = 0;
    f = fopen(nom_fichier, "r");

    while((lettre = fgetc(f)) != EOF) {
        if(lettre == '\n')
            taille++;
    }

    c.taille = taille;
    c.tab = (Ville*)malloc(taille * sizeof(Ville));

    rewind(f);

    while(!feof(f)) {
        fscanf(f, "%s %d %d", c.tab[i].nom_ville, &(c.tab[i].x), &(c.tab[i].y));
        c.tab[i].etiquette = i;
        i++;
    }

    fclose(f);

    return c;
}
```

2)Programmation génétique

La nouveauté dans ce projet que nous avons utilisé la programmation génétique. C'est une technique permettant à un programme informatique d'apprendre, par un algorithme évolutionniste, à optimiser peu à peu une population d'autres programmes pour augmenter leur degré d'adaptation (fitness) à réaliser une tâche demandée par un utilisateur.

Au niveau de son implémentation , nous avons suivi l'ordre énoncé dans le sujet , c'est à dire générer , trier , construire etc …

Voici la fonction qui génère une visite aléatoirement :

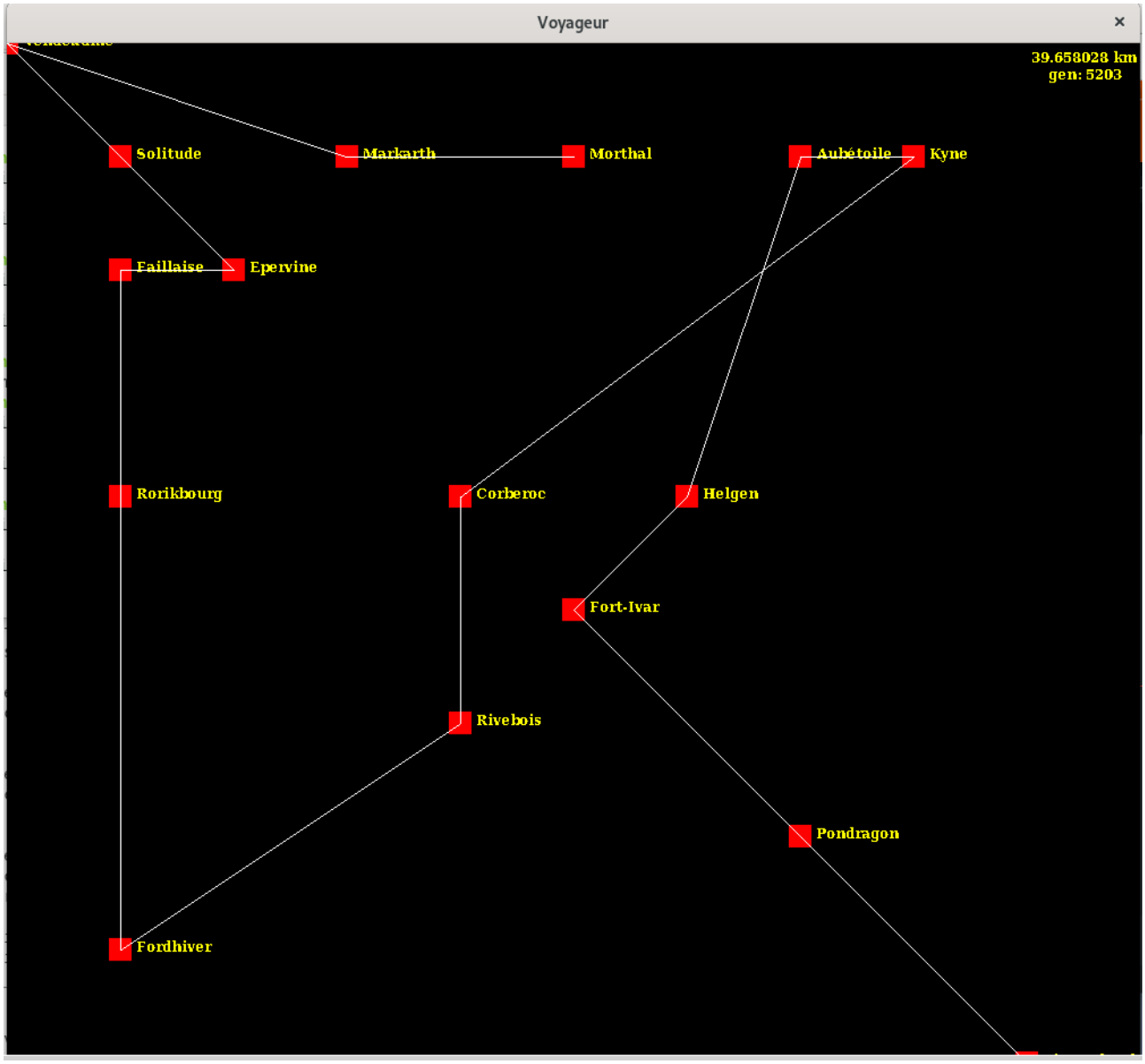
```
Visite gen_visite_alea(Carte c) {
    Visite v;
    int tab_mem[c.taille];
    int i, choix_alea;

    v = init_visite(c.taille);

    for(i = 0; i < c.taille; i++)
        tab_mem[i] = 0;

    i = 0;
    while(!(verif_tab_complet(tab_mem, c.taille))) {
        choix_alea = rand() % c.taille;
        /* Si c'est la première fois que l'on a cette étiquette. */
        if(tab_mem[choix_alea] == 0) {
            v.tab[i] = c.tab[choix_alea];
            tab_mem[choix_alea] = 1;
            i++;
        }
    }

    return v;
}
```

Enfin voici la fonction qui fait les mutations de chaque génération, c'est la fonction qui nous a posé le plus de problème dans cette partie génétique :

```c
Visite mutation(Visite v1){
    Ville res;
    int i;
    int alea_debut, alea_taille, alea_3;

    alea_debut = rand() % (v1.taille / 2);
    alea_taille = rand() % ((v1.taille - alea_debut) / 2) + 1;
    if(v1.taille - alea_debut - (2 * alea_taille) + 1 == 0)
        alea_3 = 0;
    else
        alea_3 = rand() % (v1.taille - alea_debut - (2 * alea_taille) + 1);

    /*printf("alea_debut: %d\n", alea_debut);
    printf("alea_taille: %d\n", alea_taille);
    printf("alea_3: %d\n", alea_3);*/

    /*échange*/
    for(i = alea_debut; i < alea_debut + alea_taille; i++){
        res = v1.tab[i];
        v1.tab[i] = v1.tab[i + alea_taille + alea_3];
        v1.tab[i + alea_taille + alea_3] = res;
        /*printf("%s %s\n", v1.tab[i].nom_ville, v1.tab[i + alea_taille + alea_3].nom_ville);*/
    }

    return v1;
}
```

3)Partie graphique

Au niveau de le partie graphique , nous avons décider d'utiliser la librairie MLV qui est assez facile d'utilisation par rapport au tracé entre 2 points , comparé à la librairie Ncurses. C'est donc pour cela que nous avons utilisé la librairie MLV .



On peut voir sur cette exemple , les villes sont représentés par des carré rouge (avec leur nom en jeune) , le chemin entre les ville est représenté par un trait blanc . Nous avons ajouté en haut à droite des informations sur la distance du meilleur chemin en cours et on peut aussi voir le nombre de génération faite par le programme .

Les fonctions qui nous avons utilisé pour faire l’affichage sont les suivantes :

```c
/* Affiche sur la fenêtre les villes de la Carte c. */
void affiche_carte(Carte c, int largeur, int hauteur);

/* Affiche les chemins qui lient les villes. */
void affiche_chemin(Visite v);

/* Affiche la distance de la visite en haut à droite
 * de la fenêtre. */
void affiche_distance(Visite v, int largeur, int hauteur);

/* Affiche le numéro de la génération actuel. */
void affiche_generation(int n, int largeur, int hauteur);

/* Affiche les villes et les chemins. */
void affiche_visite(Visite v, Carte c, int largeur, int hauteur);
```

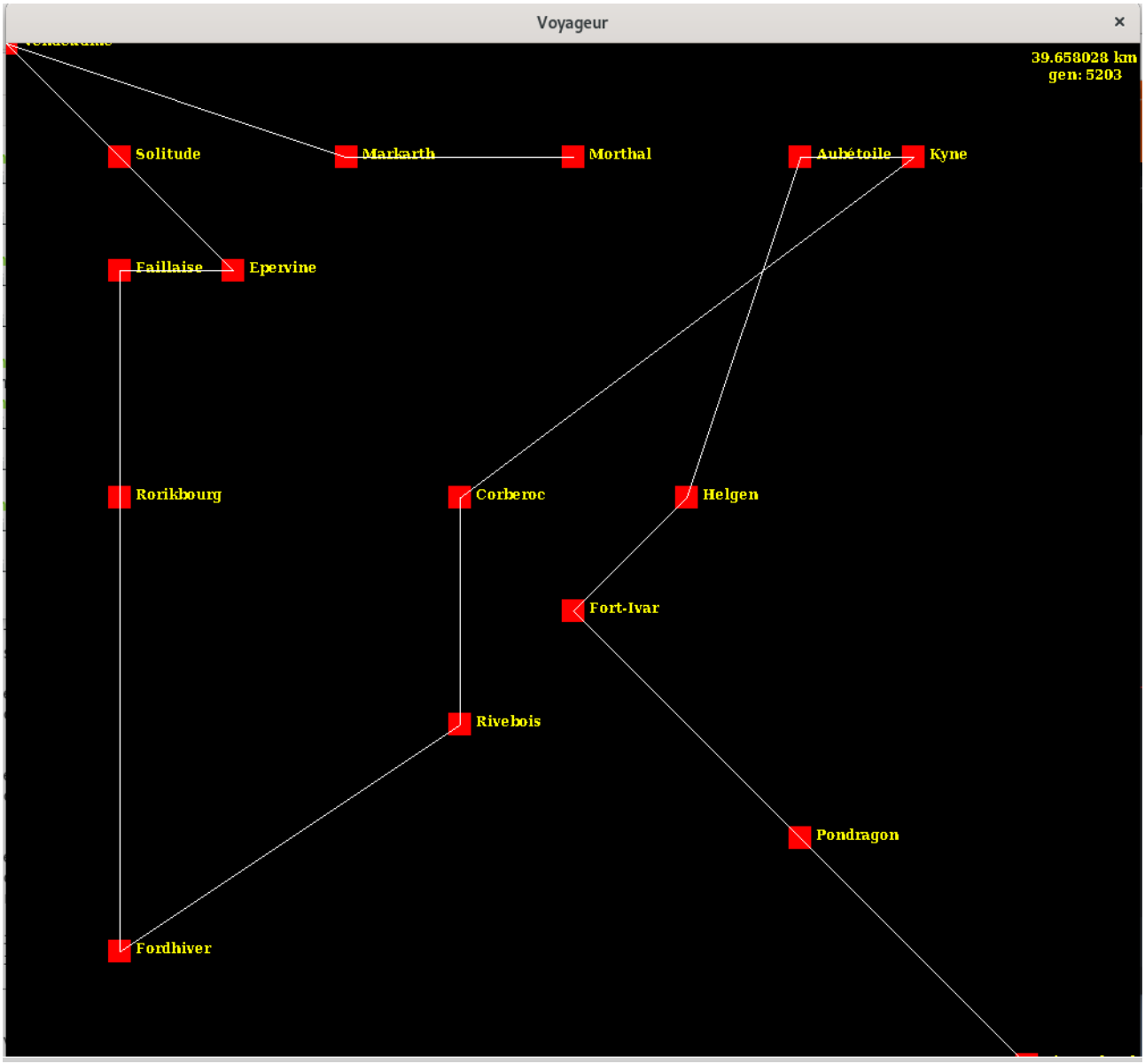
Enfin un exemple de quelque fonction :

```c
void affiche_carte(Carte c, int largeur, int hauteur) {
    int i;
    for(i = 0; i < c.taille; i++) {
        MLV_draw_filled_rectangle((c.tab[i].x + 1) * 100 - 10, (c.tab[i].y + 1) * 100 - 10, 20, 20, MLV_COLOR_RED);
        MLV_draw_text((c.tab[i].x + 1) * 100 + 15, (c.tab[i].y + 1) * 100 - 10, c.tab[i].nom_ville, MLV_COLOR_YELLOW);
    }
}

void affiche_chemin(Visite v) {
    int i;
    for(i = 0; i < v.taille - 1; i++) {
        MLV_draw_line((v.tab[i].x + 1) * 100, (v.tab[i].y + 1) * 100, (v.tab[i + 1].x + 1) * 100, (v.tab[i + 1].y + 1) * 100, MLV_COLOR_WHI
    }
}

void affiche_distance(Visite v, int largeur, int hauteur) {
    char chaine[20];
    sprintf(chaine, "%f km", distance(v));
    MLV_draw_text((largeur - 0.5) * 100 - 45, 5, chaine, MLV_COLOR_YELLOW);
}
```

Dernière remarque concernant la taille de la fenêtre , elle s'adapte au coordonnée donné par le fichier.txt contenant les villes .

4)Tests

Pour finir , à la fin de l'implémentation de notre programme , nous avons effectué plusieurs tests afin de voir le meilleur résultat que nous pouvions obtenir . Après plusieurs tests , le meilleur résultat que nous avons obtenu est une distance de ~39km en ~5000 générations avec alpha = 10 et beta = 70, et que c'est le meilleur résultat qu'on a obtenu avec ces villes .



Conclusion :

Pour conclure ce TP nous aura permis de mener à bien un projet avec une nouvelle forme de programmation , plus précisément la programmation génétique. Ce TP est beaucoup moins directif que les précédents. Il ressemble plus à une véritable spécification fournie par un client , ce qui nous a permis de le mener a notre guise et notamment par rapport à la modularisation et à la partie graphique. Au niveau des difficultés rencontrées nous avons eu des problèmes avec la partie génétique que nous avons réussi à régler mais la partie qui nous a posé le plus de problème a été la partie graphique.

Annexe 1 :

Pour compiler le programme il suffit d'aller dans le dossier TP9 contenant les fichier.c et .h , puis en ouvrant le terminal dans ce dossier on tape la commande « make » .

Annexe 2 :

Pour exécuter le programme ,on tape avec la commande ./Voyageur

Pour lancer les test on tape ./Voyageur -t

Le programme demandera alors de saisir les paramètre alpha et beta , puis il suffit d'appuyer sur entrer pour le lancer .

Pour l'arrêter , il suffit d'appuyer sur entrer aussi .