Mezouar Mikael Université Gustave Eiffel Dos Santos Loic

Rapport Projet : Algorithme des arbres

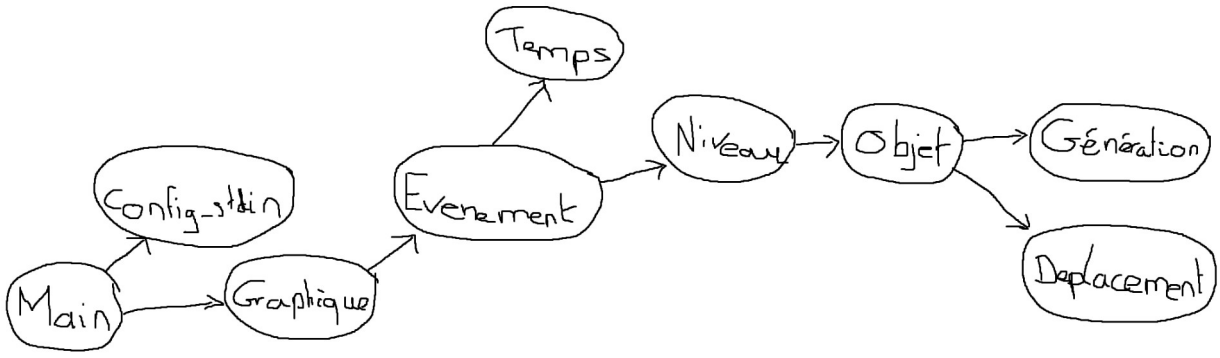
21/05/2021 L2 informatique

Introduction :

Le but du projet est d'écrire un programme réalisant un petit jeu en temps réel. Dans ce jeu des évènements se produisent et à un instant donné, on sait à l'avance à quel moment ils se produiront à l'avenir. On va utiliser une file de priorité (un tas binaire) pour stocker ces évènements. On utilisera comme clef de priorité le moment de réalisation de l'évènement: plus un évènement est proche dans l'avenir, plus il est proche de la racine du tas. Il est obligatoire d'utiliser un tas binaire pour implémenter la gestion des évènements . Le jeu consiste à déplacer un personnage sur une grille à l'aide du clavier pour l'emmener jusqu'à une destination tout en esquivant les tirs de projectile présent sur le grille. Nous allons voir dans ce rapport comment nous avons réussi à mener à bien ce projet .

Modularisation :

Pour commencer nous avons découper le projet de la sorte afin d'avoir une meilleur visibilité sur le projet et de nous faciliter le travail en binôme :



Le module main contient le Main du jeu.

Le module Objet contient des déclarations de type.

Le module déplacement et génération contiennent les déclarations de type et fonction d'initialisations.

Le module niveau qui contient le tableau d'objet , c'est à dire le tableau du jeu.

Le module évènement contient le traitement des évènement et le tas du projet.

Le module Graphique contient la partie graphique .

Le module temps permet de gérer le calcule du temps .

Détail technique :

1)Récupération du niveau

On récupère le tableau du niveau à l'aide d'un fichier texte, contenue dans le dossier du projet (dans notre cas Niveau0.niv) , ce qui permet de pouvoir changer de niveau facilement tant que le niveau qu'on nous donne est de taille conforme.

```
Coordonnees lire_taille(char* nom_fichier) {
    Coordonnees taille;
    char caractere;
    FILE *f;

    f = fopen(nom_fichier, "r");

    taille.x = 0;
    taille.y = 1;
    while((caractere = fgetc(f)) != '\n') {
        taille.x++;
    }
    taille.x--;
    while((caractere = fgetc(f)) != EOF) {
        if(caractere == '\n')
            taille.y++;
    }

    fclose(f);
    return taille;
}
```

2)Changement de la structure depl_perso_autorise :

Au niveau des changements notables que nous avons décider de faire , nous avons décidé de remplacer la structure depl_perso_autorise par une variable dans la Main.c car

on avait un problème de libération de mémoire avec la structure (erreur de segmentation) , donc nous avons décider de le mettre dans le Main.c sous forme de variable .

```
int touche, depl_perso_autorise, est_mort, gagne;
```

3)Le tas

Ensuite si l'on s'intéresse du fonctionnement liée au Tas , l'idée est la suivante :
Le tas d'évènement triée de manière croissante en fonction du moment auquel les évènements vont s'exécuter, lorsqu'un lanceur s'exécute il se retire du tas, tire entre 0 et 4 projectiles puis se rajoute avec un moment égal à maintenant + interval, un projectile se retire du tas, avance et se rajoute aussi dans le tas avec un moment égal à maintenant + allure ou rencontre un mur et se retire juste du tas.

Voici quelques fonctions qui illustrent le tas et les évènements :

```
Tas* construit_Tas(Niveau* niveau) {
    Tas* tas;
    unsigned i, j;
    Generation g;
    Evenement ev;

    tas = malloc_Tas(BLOC_ALLOC);

    for(i = 0; i < niveau->taille.y; i++) {
        for(j = 0; j < niveau->taille.x; j++) {
            if(niveau->objets[i][j].type == LANCEUR) {
                g = * (Generation *)niveau->objets[i][j].donnee_suppl;

                ev.moment = maintenant() + (g.intervalle * une_milliseconde);
                ev.coo_obj.x = j;
                ev.coo_obj.y = i;

                ajoute_evenement(tas, ev);
            }
        }
    }
    return tas;
}
```

```
Evenement ote_minimum(Tas* tas) {
    Evenement ev;
    unsigned i;

    ev = tas->valeurs[0];
    for(i = 0; i < tas->taille - 1; i++) {
        tas->valeurs[i] = tas->valeurs[i + 1];
    }
    tas->taille--;

    return ev;
}
```

```
if(un_evenement_est_pret(tas)) {
    e = ote_minimum(tas);
    execute_evenement(e, tas, niveau, &depl_perso_autorise, &est_mort);
    /*affiche_Tas(tas);*/
    affiche_Niveau(niveau);
}
/* Toujours attendre quelques millisecondes
quand on ne fait pas grand chose (ici un
seul test) dans la boucle. */
```

4)Déplacement du personnage

Pour déplacer le personnage nous avons fais de manière la manière suivante :

Lorsqu'on choisit une touche (z,q,s,d), on vérifie si c'est possible de se déplacer dans cette direction , c'est à dire vérifier si la prochaine case n'est pas un mur et si la prochaine case reste dans les bordure du niveau. Ensuite on vérifie si la case dans laquelle on veut se déplacer est vide, si c'est le cas alors on fait le déplacement , si c'est un projectile alors le personnage meurt , et si c'est la destination alors c'est gagné pour l'utilisateur .

```
void deplace_perso(Tas* tas, Niveau* niveau, int touche, int *depl_perso_autorise, int *est_mort, int *gagne) {
    Evenement e;

    switch(touche) {
        case 'z': if(niveau->coo_perso.y != 0) {
                    if(niveau->objets[niveau->coo_perso.y - 1][niveau->coo_perso.x].type == VIDE) {
                        niveau->objets[niveau->coo_perso.y - 1][niveau->coo_perso.x] = niveau->objets[niveau->coo_perso.y][niveau->coo_perso.x];
                        niveau->objets[niveau->coo_perso.y][niveau->coo_perso.x].type = VIDE;
                        niveau->coo_perso.y = niveau->coo_perso.y - 1;
                        e.moment = maintenant() + (niveau->allure_perso * une_milliseconde);
                        e.coo_obj = niveau->coo_perso;
                        *depl_perso_autorise = 0;
                        ajoute_evenement(tas, e);
                    }
                    else if(niveau->objets[niveau->coo_perso.y - 1][niveau->coo_perso.x].type == PROJECTILE)
                        *est_mort = 1;
                    else if(niveau->objets[niveau->coo_perso.y - 1][niveau->coo_perso.x].type == DESTINATION)
                        *gagne = 1;
                }
                break;
```

Guide utilisateur :

Lorsqu'on lance le jeu avec les commandes disponible à la fin du rapport (annexe 1 et 2) , l'utilisateur a le choix entre la version graphique et une version terminal.

```
(base) padawan@DebianUPEM2019:~/Téléchargements/Projet_4$ ./tempsreel
Version Terminal ou Version Graphique(T ou G):
```

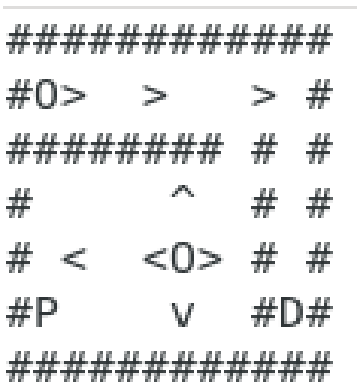
Il faut entrer T pour la version terminal ou G pour la version Graphique puis entrer pour lancer le jeu.

Lorsque l'on est sur une des deux versions , il suffit juste de déplacer le Personnage (P) jusqu'à la destination (D) à l'aide des touches z (vers le haut) , q (vers la gauche) , s (vers le bas) et d (vers la droite) . Attention à ne pas spammer les touches trop vite , en effet le jeu est conçu avec un temps de latence de 250 ms entre 2 touches (c'est à dire que lorsqu'on appuie sur une touche , le personnage se déplacera et on pourra alors redéplacer le personnage 250 ms après avoir appuyer de la touche juste avant) , toute les touches faites dans cet intervalle de temps seront ignorés.

Version graphique :

Au niveau de la version graphique , étant donner que nous avons travaillé durant tout le semestre avec Ncurses en Perfectionnement en C , nous avons décider avec l'accord d'un professeur de faire une version graphique avec Ncurses ce qui nous a permis d'être directement à l'aise avec l'implémentation de la version graphique et surtout d'éviter de devoir se redocumenter grâce à nos acquis du Perfectionnement en C .

Voici un exemple en image de la version graphique sur Ncurses , qui ressemble à la version terminal mais celle-ci est beaucoup plus adapter au gameplay et surtout plus agréable lorsque l'on joue.



Conclusion :

Pour conclure ce projet nous a permis de mettre en application direct la notion de tas binaire , il aura été très bénéfique dans la compréhension de la notion en vu de l'examen . Au niveau du fonctionnement et de la répartition du travail , nous avons travaillé en appel discord avec des partages d'écrans simultanés, ce qui nous a permis de travailler ensemble un maximum , le fait que le projet soit guidé nous aura aussi beaucoup aidé. Néanmoins nous avons quand même rencontré des difficultés au cours de ce projet et notamment avec l'allocation dynamique à faire sur les différents éléments du projet ou bien nous avons du résoudre énormément d'erreur de segmentation qui étaient des erreurs compliquer à identifier dans l'immédiat.

Annexe 1 :

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

Quand on tape make test_config_stdin , puis ./test_config_stdin. Ces 2 commandes permettent de voir si l'entrée standard fonctionne bien .

Annexe 2 :

Notre projet se compile avec les commandes en annexe 1 (la première) , puis on l'exécute avec la commande ./tempsreel.