LAMOUR Lauriane
ADOUANE Cylia

Compte-rendu du projet « Bataille navale »

1 - Explication du projet

Nous devions coder un jeu de bataille navale sous la forme d'un MMO. Le déroulement du jeu est similaire à la bataille navale sauf que chaque bateau représente un joueur. Deux modes devait être coder : le mode "*seul contre tous*" et le mode "*équipe vs équipe*".
Nous avons opté pour affichage en log.

2 - Carte navale et gestion du jeu

La carte navale et les paramètres de jeu sont initialisés par la lecture en entrée du nom du fichier contenant tous ses paramètres. Pour récupérer ses paramètres, nous avons créer la fonction suivante : ***char* lecture_fichier(const char*chemin)***.
Elle prend en entrée le chemin du fichier, lit dans le fichier et récupère son contenu dans un tableau de caractères qui est renvoyé.

Le fichier contient des données numériques tels que la taille de la carte navale ou le nombre de joueurs par exemple. Or nous récupérons le contenu du fichier sous forme d'un tableau de caractères. Ainsi pour récupérer les données numériques sous forme d'un entier, on a créé les fonctions : ***int chartoint(char*t,int i)*** et **int decalage (int a)**.
La première fonction prend en argument le tableau de caractères contenant le contenu du fichier et l'indice de lecture dans le tableau. On récupère la donnée numérique sous forme d'un entier. La seconde fonction calcule à quel indice la lecture du tableau en est.

Les paramètres de jeu sont initialisés par la fonction ***void init_param (navalmap_t*m,int*nb_tours,int *coque,int *kerosene,char*sauv)*** qui prend en argument un pointeur de type navalmap_t permettant de récupérer la taille de la carte, le nombre de joueurs et le type de la carte, l'adresse d'un entier permettant de récupérer le nombre de tours, celle d'un deuxième entier pour récupérer le nombre de coque, celle d'un troisième entier pour récupérer le nombre de kérosène et le tableau contenant le contenu du fichier.

Afin de gérer les bateaux tout au long de la partie, nous avons créé une structure appelée SHIP qui prend en champ l'identifiant, la coque et le kérosène du bateau et

ensuite, on a créé un tableau de struct SHIP afin de facilement récupérer l'ensemble des données de chaque bateau.
L'initialisation de l'état de chaque bateau se fait grâce à la fonction ***void init_ship(SHIP*s,int kerosene,int coque,int nb_ship)*** qui prend en argument le tableau réunissant l'état de tous les bateaux, le kérosène et la coque de départ, et le nombre de joueurs.

3 - Mode seul contre tous

Dans le mode *seul contre tous*, le serveur est représenté par le processus père qui va créer un nombre processus fils dépendant du nombre de joueurs. Le processus père envoie à chaque processus fils, l'état de son bateau et chaque processus fils envoie son choix d'action au processus père. Le processus père attend que tous ces fils ont fait leur choix, récupère leur choix grâce à un tube anonyme et les applique selon leur catégorie.

Nous avons créé une structure *JOUEUR*, qui va permet de regrouper les données propres et nécessaire au choix de chaque joueur. Cette structure est composée de l'état du bateau (coque,kérosène), du tour auquel on se trouve, les coordonnées de l'ennemi le plus proche, et une structure *CHOIX*, qui permet de sauvegarder le choix du joueur, la catégorie du choix et l'identifiant du joueur ayant fait son choix.

Le choix du joueur dépend d'un algorithme de décision mis en place dans la fonction **CHOIX choix_du_joueur(navalmap_t*n,JOUEUR J)** dont on récupère le choix du joueur. On définit l'ordre d'application des actions grâce à la fonction **JOUEUR * ordre_de_passage(navalmap_t *n,JOUEUR *J,int nb_joueurs)** qui trie les actions selon leur catégorie. Enfin on applique le choix de chaque joueur grâce à la fonction **void application_choix(JOUEUR *ordre,navalmap_t*n,SHIP*s,int i)**.

La mort des bateaux est gérée par la fonction **void verif_etat_bat(navalmap_t *m,SHIP *s,JOUEUR*J,int *cmp_bat)** qui regarde si un bateau a sa coque ou son kérosène à 0 et va enlever le bateau de la carte en faisant les modifications nécessaires.
Les conditions de victoires dépendent du nombre de joueurs restants. Ce nombre est calculé par la fonction **int nb_joueurs_restants(navalmap_t*n,JOUEUR *J)**.
On désigne le vainqueur avec la fonction **void gagne(navalmap_t*m,SHIP*s,int nb_bat)** qui donne le joueur gagnant si au bout du nombre de tours donné, il y en a un sinon déclare l'égalité.

4) Mode equipe vs equipe

Le mode *equipe vs equipe* dépend d'un serveur représenté par le processus père qui va créer un nombre de processus fils dépendant du nombre d'équipe donné par le fichier d'entrée. Nous avons décidé que chaque équipe sera composé de deux joueurs. Ainsi, chaque processus fils va créer 2 threads représentant les joueurs. Le processus père envoie à chaque processus, les données de ses joueurs. Ensuite, chaque joueur de chaque équipe fait son choix, le serveur récupère l'ensemble des choix grâce à un tube anonyme et applique les choix selon leur catégorie.

Pour gérer l'algorithme de décision, nous avions besoin de récupérer toutes les données propres à chaque joueur de chaque équipe. Donc nous avons créé une structure *Joueur* qui prend en champ :

- le numéro de l'équipe
- le tour auquel le jeu en est
- les coordonnées de l'ennemi le plus proche
- le choix du joueur
- l'état du bateau (coque/kérosène)
- la carte navale
- un mutex

L'algorithme de décision est lancé par la fonction **void* choice(void *a)**. Il est protégé par un mutex afin d'éviter de récupérer le choix de l'autre thread.
Lorsque tous les choix ont été fait, la fonction **CHOIX * ordre_des_actions (navalmap_t *n,Joueur**J,int nb_joueurs)** tri les actions selon leur catégorie et la fonction **void apply_choice(CHOIX *c,Joueur**J,navalmap_t*n,SHIP*s,int nbE,int nbJ,int cmpJ,int tour)** applique les choix.

La gestion de la partie se fait par la fonction **Joueur**verification_etat (navalmap_t *m,SHIP *s,int *cmp_bat,int *nb_equipes,Joueur **J)** qui supprime les bateaux coulés, des fonctions **int nbJoueursRestants(navalmap_t *n,Joueur**J)** et **int nbJoueursRestantsParEquipe(navalmap_t *n,Joueur**J,int idE)** qui calcule le nombre de joueurs et d'équipes restants et de la fonction **void gagne_equipe (navalmap_t*m,int*nb_bat,int cmp_bat)** qui désigne l'équipe gagnante si il y en a une.

Dans le cas des actions de radar, ce mode de jeu a obligé à restreindre le champ de recherche des radars afin d'éviter qu'ils ne détectent comme bateau ennemi, des bateaux de leur propre équipe.
La fonction **SHIP *recherche(Joueur **J,SHIP *s,int idBat,int id_equipe,int nb_joueurs,int nb_equipes,int*cmp_bat)** permet de restreindre ce champ car nos deux fonctions dépendent du tableau de l'état des bateaux. Ainsi, on recrée un tableau de l'état de bateaux ne contenant que les bateaux des autres équipes.
Les fonctions **void mise_a_jour_radar_global(navalmap_t *n,Joueur **J,SHIP*s,int idE,int nbJ,int nbE,int icase)** et **void mise_a_jour_radar_local**

(navalmap_t *n,Joueur **J,SHIP*s,int idE,int nbJ,int nbE,int icase) dépendent de ce nouveau tableau créé et met à jour le tableau global d'état des bateaux.

Conclusion

Nous avons connu quelques difficultés durant ce projet car il était très difficile de partir d'un code que l'on avait dû mal à comprendre au début. Lorsqu'on pensait avoir compris ce code, on commençait à coder puis on se rendait compte de quelques bugs dans notre code ce qui nous obligeait à relire le code donné à nouveau pour essayer de voir ce qu'on n'avait pas compris.
De plus, le fait que le sujet ne soit pas donné en entier était préjudiciable car lorsqu'on avait fini une partie, on ne pouvait pas continuer donc on attendait que la suite du sujet sorte et on laissait le projet de côté. Donc lorsque l'on reprenait le projet, on était un peu perdu.

Malgré ces difficultés, ce projet a été très enrichissant. Cela nous a permis d'apprendre à travailler en équipe, ce qui n'est pas simple à gérer. De plus, cela nous a aidé à mieux comprendre le fonctionnement des processus et des threads ainsi que l'importance du partage de données et de la synchronisation.