COMPTE RENDU
SYSTEME D'EXPLOITATION

SERHAN Wissam
CRENIER Melvin

Partie I :

En premier lieu, nous avons eu du mal à savoir comment lire le fichier d'entrée et les structures de données de la librairie de base ne permettait pas de stocker la totalité des informations. Pour sortir de ce problème, nous avons décidé de personnaliser la librairie en y ajoutant des structures de données.

En ce qui concerne la méthode d'affichage, nous avons choisi l'affichage littéral.

Partie II :

Dans cette partie, nous avons eu des difficultés à visualiser comment créer le radar. Nous sommes donc passé à la partie suivante et avons donc procédé à la mise en place des mouvements. Nous n'avons rencontré aucun problème concernant les mouvements.

Ensuite, nous sommes arrivés au codage des ATK. A partir de ce moment, nous étions obligé de retourner sur le codage du Radar pour obtenir la position adverse avant de pouvoir attaquer.

Pour coder le radar nous avons donc établi une structure Radar récupérant toutes les informations nécessaires concernant le bateau détecté, le temps de vie du radar et surtout la distance avec le vaisseau adverse. La distance nous a permis d'élaborer le mouvement automatique en direction du vaisseau ennemi.

Il nous a donc aussi permis de coder l'ATK en fonction de la position adverse et donc de le toucher.

Tous les choix possibles (ATK,MOV,RADAR) se trouvent donc dans l'algorithme de décision. Notre algorithme de décision tourne autour du Radar. En effet, les mouvements et attaques ne s'effectue qu'au dépend de la position de l'adversaire, il n'y a pas de facteur chance dans notre algorithme. C'est pour cela que le Radar est indispensable dans notre algorithme. Pour que le vitesse du jeu soit rapide, nous avons choisi un vieillissement de radar valant 1 tour.

En ce qui concerne les mouvements, nous avons choisi de nous rapprocher du vaisseau adversaire de cette façon :

-Soit X et Y les coordonnées de la distance entre le bateau du J1 et celui du J2 :
  -Si X<Y, on se déplace de 2 sur l'axe Y
  -Si Y<X, on se déplace de 2 sur l'axe X

-Si X=Y, on se déplace de 1 sur l'axe X et 1 sur l'axe Y (On se déplace de 2 en diagonale.

A ce moment, nous avons rencontrer un problème au niveau des mouvements. Nous avons donc mit un peu de temps à comprendre d'où il provenait et il s'avérait que le souci était simplement une inversion des X et Y.

Après avoir élaboré notre algorithme de décision, notre game-over et le main, le programme fonctionnait correctement mise à part le fait que les valeurs des coques pouvait tomber en dessous de 0 sans que la partie ne s'achève. Nous avons résolu ce problème en ajoutant un test de mort dans le cas où tout les navires sont morts. Cela nous a permis de pouvoir exécuter le programme en ayant un arrêt au moment où l'un des deux navires, ou les 2 navires meurent.

Mais nous avons rencontré un autre problème en résolvant ce dernier. En effet, les deux navires effectuent les mêmes actions. Il arrive donc un moment où les navires s'entre-tuent mais sachant que le Joueur 1 joue avant le 2, le 2 devrait perdre. Cependant, le serveur récolte les informations de décisions au début du tour. Les joueurs 1 et 2 envoient donc leur information indiquant l'attaque et le Joueur 2 envoie donc son attaque malgré que la coque tombe sous la barre des 0. Pour résoudre ce problème nous avons décidé d'ajouter une variable int en_vie prenant comme valeur 1 ou 0 pour nous permettre de tester directement quand le vaisseau coule. A partir de cela nous avons dû remplacer une grande partie de nos test concernant la mort des navires.

Partie III :

La partie sur les équipes fonctionne sur le même principe que la partie 2. En effet, on créer les équipes et les gère comme des joueurs mais en groupe grâce à une structure gérant les équipes. En effet notre structure contient un ID pour chaque équipe et ainsi qu'un nombre de joueurs. Ensuite, nous gérons les décisions de chaque joueur de chaque équipe comme dans la partie 2.