Projet de Programmation en C : Bataille navale

Nicolas Dos Santos Théo Julien Maxime Pegane Vingadas

8 décembre 2019

Table des matières

I Problème demandé

Le projet est simple, développer en C un jeu permettant de jouer à la "Bataille navale". Le jeu oppose deux joueurs, soient humains soient contrôlés par un ordinateur. Les deux joueurs doivent créer une grille secrète de navires, puis chacun essaie de faire couler la flotte de l'ennemi.
Quelques règles nous ont été imposés :

- La taille de la grille peut être variable, numérotée horizontalement et lettrée verticalement.
- La taille et le nombre de bateaux sont choisis par les joueurs.
- Chaque joueur possède le même nombre de bateaux, de taille égale.
- Si la "bombe" touche un bateau, le joueur continue à tirer.
- Si le tir du joueur 1 échoue, c'est le tour du joueur 2 de tirer.
- Pour faire couler un bateau, il faut avoir torpillé les coordonnées de toutes les cases que composent le bateau.
- Le jeu s'arrête dès que la flotte entière d'un joueur a coulé.

II Cahier des charges

A la base, le programme doit permettre de jouer pleinement à la Bataille navale à deux :

- Humain contre humain,
- Humain contre l'ordinateur (qui, dans la version la plus basique du programme, jouera un coup au hasard). Il faudra ensuite optimiser votre programme, notamment en implémentant un joueur artificiel digne de ce nom.

Voici une liste des fonctionnalités demandées :

II.1 Brique logicielle de base

- **Initialisation :** Il faut pouvoir placer un certain nombre de navires sur un plateau de jeu, de façon la plus paramétrable possible (en particulier, on doit pouvoir spécifier la taille de la grille, le nombre de navires à placer et le nombre de cases qu'ils prennent, sachant qu'il peut y avoir plusieurs types de navires). Attention à respecter le couloir d'eau entre les bateaux.

- **Tir :** Il faut pouvoir déterminer le résultat d'un tir sur une certaine case du plateau de l'autre joueur. Le tir peut être raté (viser une case d'eau), toucher un navire ou le couler, voire couler le dernier navire adverse et faire gagner la partie. Attention à prévoir les tirs hors plateau.

- **Demande de tir au joueur artificiel :** Il faut pouvoir demander à l'ordinateur de choisir les coordonnées de la case sur laquelle il souhaite tirer. Dans un premier temps, le choix est complètement aléatoire. Nous optimisons cette fonction par la suite.

- **Affichage textuel d'une grille :** Il est intéressant, du moins pour des raisons de débogage, de pouvoir visualiser le plateau de jeu.

- **Ordonnanceur :** Il faut pouvoir jouer à la Bataille Navale de bout en bout, avec un menu textuel, des options de jeu, et une boucle qui demande au joueur courant sa prochaine action avant de passer à l'autre, jusqu'à la victoire de l'un d'entre eux.

II.2 Fonctions avancées

- **Optimisation du joueur artificiel :** Cette fois, l'IA ne doit plus jouer un coup au hasard, mais «réfléchir» à la case sur laquelle il faut tirer en fonction de plusieurs paramètres.

- **Enregistrement/chargement d'une partie :** Il faut pouvoir sauvegarder l'état d'une partie dans un fichier texte, afin de pouvoir y revenir plus tard. L'option de chargement d'une partie doit donc être incluse dans le menu.

- **Compilation Makefile :** Il faut pouvoir exécuter rapidement le programme.

- **Score des joueurs :** Il faut pouvoir évaluer le nombre de tentative réalisée réussi ou ratée par le joueur et en fonction, mettre à jour un score.

- **Amélioration du menu :** Il y a toujours moyen d'optimiser l'esthétique, et surtout l'ergonomie, de votre menu. Utilisation du clavier, de la souris, jeu sur interface graphique, sprites, sons, animations, cinématiques...

- **Variantes de la Bataille navale :** Il y en a beaucoup

III Description du programme

Avant la description précise de notre programme, nous vous informons que nous avons commenté toutes nos fonctions. Ainsi, si vous voulez plus de précision sur les fonctions de notre programme, vous pouvez directement regarder le code de celui-ci. Tout d'abord, nous avons sept fichiers :

- Trois fichiers d'entête : FcBasique.h, FoncCurse.h, fonction.h. Nous décrirons dans ce rapport seulement FcBasique.h puisque les deux autres fichiers entête n'ont pas de spécificités particulières.

- Quatre fichiers programme : FcBasique.c, FoncCurse.c, fonction.c et programme.c que nous allons décrire dans ce rapport.

III.1 FcBasique.h

Le fichier d'entête FcBasique.h sera utilisé par tous les autres fichiers. Nous avons donc importé dans ce fichier les bibliothèques de base : stdio.h, unistd.h, stdlib.h, ctype.h, string.h, time.h, math.h, assert.h et ncurses.h .
Puis, nous avons implémenté les structures de base :

Struct boat

La structure boat permet de définir la structure d'un bateau, elle est constitué de trois éléments :

- La liste tint_coor de une dimension et de deux cases permettant de stocker les coordonnées de la case la plus en bas à gauche du bateau.

- L'entier int_size représentant la taille du bateau.

- L'entier int_dir qui permet de savoir la direction du bateau : la valeur est 0 si le bateau est vers la droite, 1 si le bateau est vers le haut.

Struct state

La structure state permet de définir la structure d'une case, elle est constituée de trois éléments :

- L'entier int_tir qui indique si la case a déjà été ciblé ou non. Si oui, elle prend la valeur 1, sinon elle prend la valeur 0.

- L'entier int_hit qui indique l'état d'une case qui a été tiré. S'il n'y a pas de bateau sur la case, int_hit prend la valeur 0, sinon si le bateau est coulé, il prend la valeur 2 et sinon il prend la valeur 1.

- La structure boat bateau, qui est décrite ci-dessus.

Struct tab

La structure tab permet de définir la structure d'un tableau de trois dimensions,représentant le plateau de jeu de deux joueurs, d'où la troisième dimension. Elle Les règles et conditions ont été

respectés ce qui nous a vallut un week-end bien chargé en codage est constituée de cinq éléments :

- La structure state*** ppstate_grid, structure d'un tableau tridimensionnel. La structure a été décrite ci-dessus.

- L'entier int_wid représentant la largeur du tableau.

- L'entier int_dir représentant la longueur du tableau.

- L'entier int_IA qui prend la valeur 1 s'il y a une intelligence artificielle, sinon il prend la valeur 0.

- La liste tint_nombreBateau[3], qui nous permettra d'afficher le score. La première case de la liste contient le score du joueur 1, la deuxième case contient le score du joueur 2 et la dernière case correspond au nombre total de case contenant un bout de bateau.

III.2 FcBasique.c

Le fichier FcBasique.c est très utile puisqu'il regroupe toutes les fonctions qui nous permettrons d'avoir un meilleur affichage.

int saisieEntier

La fonction saisieEntier ne prend rien en argument, elle permet de vérifier la saisie de l'utilisateur et retourne un entier.

void clrcsr

La procédure clrcsr permet d'effacer le contenu du terminal, ce qui est pratique pour notre jeu puisque les joueurs ne doivent pas voir le tableau de jeu adverses.

void resetColor

La procédure resetColor permet de remettre la couleur par défaut.

void ColorBlue

La procédure ColorBlue permet de modifier la couleur en bleu.

void ColorRed

La procédure ColorRed permet de modifier la couleur en rouge.

void ColorMag

La procédure ColorMag permet de modifier la couleur en magenta.

III.3 FoncCurse.c

Nous n'allons pas entrer dans les détails des fonctions de ce fichier puisque les fonctions int is_blod, void init_colorpairs, short curs_color, int colonum, void setcolor et void unsetcolor nous servent à faciliter la définition des couleurs de l'écriture et du fond : foreground et background. Les autres fonctions nous facilitent l'affichage du plateau et des légendes de celui-ci.

III.4 fonction.c

Le fichier fonction.c est le centre du programme. En effet, il contient toutes les fonctions qui constitue notre programme.

struct tab init

La fonction init prend en argument une structure tab. Elle permet d'initialiser un tableau tridimensionnel. Tout d'abord, elle demande à l'utilisateur le nombre de colonnes et le nombre de lignes du plateau de jeu puis alloue de la mémoire pour ce tableau. Ensuite, la fonction initialise tous les éléments de la structure tab à 0 et renvoie le tableau.

void AffichageTir

La procédure AffichageTir prend en argument une structure tab, et trois entiers : le joueur actif et deux entiers représentant les coordonnées d'une case. Selon la valeur de int_hit de cette case dans le tableau représentant les cases tirées ou non du plateau du joueur adverse, la procédure affiche cette case d'une couleur particulière : magenta si le bateau est coulé, rouge si le bateau est touché, bleu si le tir est tombé dans l'eau et vert si la case n'a jamais été ciblé.

void Affichage

La procédure Affichage prend en argument une structure tab, le joueur actif et un caractère. Cette procédure permet l'affichage du plateau en fonction du caractère prit en argument :

- si le caractère est 'b', le programme affiche le plateau avec nos bateaux.

- si le caractère est 't', le programme affiche le plateau où nous tirons grâce à la fonction AffichageTir, avec tous les tirs déjà effectués.

- si le caractère est 'a', le programme affiche nos bateaux avec les tirs ennemis.

int verifRectangle

La fonction verfiRectangle prend en argument une structure tab, le joueur actif et quatre entiers représentants des coordonnées. Cette fonction permet de vérifier si le joueur peut poser un bateau à l'endroit souhaité en prenant en compte les couloirs d'eau obligatoire entre chaque bateau. La fonction renvoie 0 si les conditions sont remplies et 1 s'il y a une erreur.

int def_x_y

La fonction def_x_y prend en argument une structure tab, le joueur actif, les coordonnées de la case là plus en bas à gauche du bateau, la direction et la taille du bateau. Cette fonction permet de vérifier si toutes les cases formant le bateau sont dans le tableau, si oui, elle utilise la fonction verifRectangle pour confirmer que le bateau peut être placé à l'endroit souhaité dans le tableau. La fonction renvoie 0 si les conditions sont remplies et -1 s'il y a une erreur.

struct tab placement

La fonction placement prend en argument une structure tab, le joueur actif et la taille du bateau que le joueur va placer. Cette fonction permet de demander à l'utilisateur où veut-il placer ses bateaux grâce à un affichage qu'il peut contrôler avec les flèches du clavier. En effet, le programme va afficher un bateau de la taille prise en argument (il sera toujours initialisé en haut à gauche du plateau, à l'horizontale), le joueur placera son bateau avec les flèches et la touche 'Entrée' du clavier et pourra changer la direction du bateau grâce à la touche 'espace'. L'emplacement du bateau sera vérifié avec la fonction def_x_y, s'il est autorisé, le bateau sera affiché en vert, sinon il sera afficher en rouge et la fonction exécutera un bruitage 'beep' si le joueur essaye de placer un bateau hors du plateau de jeu. Lorsque le bateau est placé, la fonction affiche 'Bateau placé !' au milieu du terminal. Pour finir, la fonction renvoie la structure tab contenant les bateaux placés.

void verifDimensionBash

La procédure verifDimensionBash vérifie si les dimensions du terminal permettent l'exécution du jeu. La fonction affiche les dimensions actuelles du terminal et si celles-ci sont trop petites, la fonction demandera à agrandir le terminal.

struct tab CreaBateau

La fonction CreaBateau prend en argument une structure tab, elle permet de demander avant de commencer la partie :

- La taille du plus grand bateau
- Le nombre de bateaux souhaité selon la taille du bateau

Attention, si la taille du plus grand bateau est supérieure à la taille du plateau de jeu, la saisie est redemandée. Ensuite la fonction vérifie si le terminal supporte les couleurs utilisées, si non, la fonction fait quitter le programme. Puis s'il y a une intelligence artificielle qui joue, la fonction demande si nous voulons afficher les placements de celle-ci et finit en renvoyant la structure tab.

struct tab touchercouler

La fonction touchercouler prend en argument une structure tab, le joueur actif et les coordonnées de la case tiré par le joueur. Cette fonction permet de vérifier si le bateau est coulé ou non. Si le bateau n'est pas coulé, la fonction revoie la structure tab prise en argument ; alors que si le bateau est coulé, la fonction modifie la valeur de int_hit des cases constituant le bateau à 2 afin d'afficher le bateau en magenta et renvoie la structure tab modifiée.

struct tab viserTirer

La fonction prend en argument une structure tab et le joueur actif. Elle est basée sur le même principe que la fonction placement : avec la fonction AffichageTir et les flèches du clavier, le joueur choisit la case sur laquelle il souhaite tirer puis appuie sur 'Entrée' pour tirer. La couleur de la case varie selon si la case a déjà été tirée ou non et s'il y a un bateau touché ou coulé sur les cases tirées. Le curseur est initialisé par défaut sur la case là plus en haut à gauche du tableau et s'affiche en rouge si le joueur essaye de tirer sur une case déjà tirée, qu'il repère facilment grâce aux différentes couleurs, ou s'affiche en vert s'il est autorisé à tirer dans cette case. Si le joueur essaye de tiré en dehors du plateau de jeu, la fonction exécute un bruitage 'beep'. En plus des indications habituelles, la fonction affiche 'Partie non sauvegardée' ou 'Partie sauvegardée' selon l'état de la partie. Ainsi à cette étape, le joueur peut sauvegarder la partie en appuyant sur la touche 's' du clavier et il peut la quitter grâce à la touche 'q' ; un message de vérification 'Etes-vous sûr ?(o/n) : ' s'affichera afin d'éviter de quitter la partie involontairement. La dernière étape de cette fonction est un autre affichage, la fonction affiche :

- 'BOOUUM !' si le joueur touche un bateau
- 'GIGABOOUM !' si le joueur coule un bateau
- 'PLOUF !' si le tir du joueur tombe dans l'eau

La fonction viserTirer utilise donc la fonction toucherCouler pour savoir l'état de la case sur laquelle le joueur à tiré. Le joueur peut retirer s'il a touché un bateau mais la partie se termine si le joueur a détruit tous les bateaux adverses. Pour finir, la fonction renvoie la structure tab modifiée.

int ecriture

La fonction ecriture prend en argument une structure tab et le joueur actif. Elle initialise une liste de sept cases par :

- Case 1⟶la largeur du tableau
- Case 2⟶la longueur du tableau
- Case 3⟶le score du joueur 1
- Case 4⟶le score du joueur 2
- Case 5⟶le nombre total de case contenant un bout de bateau
- Case 6⟶le joueur actif
- Case 7⟶int_IA, indication si une intelligence artificielle joue

Elle permet de sauvegarder toutes ces informations en binaire dans un fichier. Pendant la sauvegarde, si le fichier n'a pas pu s'ouvrir ou qu'il y a une erreur lors de l'écriture du tableau, la fonction affiche un message d'erreur et quitte le programme. S'il n'y a pas d'erreur, le programme continue.

struct tab lecture

La fonction lecture prend en argument un pointeur qui représente le joueur qui est le prochain à jouer. La fonction déclare un tableau de sept cases qui prendra comme valeur les informations stockées dans le fichier. Ensuite, la fonction, à l'inverse de la fonction écriture, affecte les paramètres du tableau à nos variables générales pui son alloue la mémoire necessaire pour notre tableau tridimensionnel. S'il y a des erreurs lors de la lecture du tableau, la fonction nous sort du programme avec un message d'erreur.

bataillenavale

La fonction bataillenavale est la fonction qui met en relation les autres fonctions. Si une sauvegarde la partie existe, la fonction demande aux joueurs s'ils veulent la charger. Si oui, bataillenavale lance les fonctions lecture et verifDimensionBash afin de vérifier qu'il n'y ait pas d'erreurs de lecture ou de dimension du terminal. S'il n'y a pas de partie sauvegarder ou que les joueurs veulent commencer une nouvelle partie, la fonction demande s'il y a une intelligence artificielle, si oui à quel niveau de difficulté. Ensuite il y a le lancement du jeu : tant que le score du joueur 1 et le score du joueur 2 sont inférieurs au nombre total de case contenant un bout de tableau : s'il y a une intelligence artificielle, bataillenavale lance la fonction IA_TIRE sinon elle lance la fonction viserTirer puis change de joueur. Puis la fonction affiche le joueur gagnant, désalloue la mémoire du tableau et affiche une nouvelle fois le gagnant pour s'assurer que les joueurs n'aient pas passé trop vite l'affichage du gagnant. Pour finir, la fonction renvoie la constante 'EXIT_SUCCESS' indiquant que la fonction s'est bien exécuté.

int* IA_PROBA

La fonction IA_PROBA est le coeur de notre intelligence artificielle, c'est son centre de décision, il prend en argument un tableau d'entier qui sera notre 'tableau de probabilités'. En effet ce tableau sera remplie d'entier positif, on indexe les coordonnées des cases dont la valeur est non-nulle dans ce tableau, on choisit un entier entre 0 et le nombre de cases non-vide, puis on retourne la coordonnée à l'index correspondant. Cette manière de 'réflechir'de l'intelligence artificielle est assez flexible ce qui lui permet d'être utilisée dans les fonctions de placement de bateau et les fonction de visé et de tir.

int* IA_PLACEMENT

La fonction IA_PLACEMENT nous donne une coordonnée en fonction d'une structure tableau et d'un entier taille. La fonction créée d'abord un tableau de probabilité, il met à 0 les endroits correspondant à une case où on l'on ne peut pas placer de bateau, à 1 les endroits où l'on peut placer le bateau uniquement à l'horizontale, à 2 s'il est uniquement plaçable verticalement et à 3 s'il est plaçable dans les deux sens. On utlise ensuite IA_PROBA pour choisir une de ces cases, on stocke la direction en plus des coordonnés dans une même variable avant de retourner cette variable.

struct tab IA_PLACE

La fonction IA_PLACE Permet de placer les bateau en fonction de leur taille dans une structure tableau, on commence d'abord par génerer des coordonnées qui se présente dans une liste de 3 cases : abscisse, ordonnée, direction. On procède ensuite dans une boucle for avec un nombre d'itération égal à la taille voulue du bateau. Dans cette boucle on indique sur chaque case du bateau : les coordonnées de la case principale du bateau (celle la plus en bas à gauche), la taille du bateau et sa direction. On retourne ensuite ce tableau.

int RIA_VISE**

La fonction RIA_VISE retourne des coordonnées de tir aléatoire en fonction d'une structure tableau, on créée un tableau de probabilités selon la règle suivante : les case sur lesquelles l'intelligence artificielle a déjà tiré sont mises à 0 les autres à 1. On utilise IA_PROBA pour choisir les coordonnées d'une de ces cases, puis on retourne ces coordonnées.

int* IA_FINE

la fonction IA_FINE permet de créer un tableau de probabilités plus *fin* par rapport aux coordonnées d'une case du tableau que l'on sait touchée, donc où se trouve un bateau. on regarde dans chaque direction (Haut, bas, gauche, droite) tout d'abord si la case existe afin d'éviter l'erreur de segmentation, puis si l'on y a déja tiré, si c'est le cas on laisse cette case à 0, si on y a jamais tiré on met cette case à 1, puis on retourne ensuite ce tableau.

int* IA_VISE

la fonction IA_VISE permet de retourner des coordonnées choisi et non complètement aléatoire , on créée d'abord un tableau d'entier ayant leur cases correspondant à un tir réussi à 1 et les autres à 0. On utilise ensuite IA_PROBA pour choisir l'une de ces cases, pour ensuite appeler la fonction IA_FINE avec les coordonnées de cette case et obtenir un tableau de probabilité. On réutilise ensuite IA_PROBA pour choisir une case dans le tableau de probabilité et retourner ses coordonnées.

struct tab IA _TIRE

La fonction IA_TIRE permet de mettre à jour le tableau apres avoir choisi des coordonnée de tir. On a en argument une structure tableau et un entier *difficulté* qui permet de savoir quelle methode va être choisie pour viser une case. Selon la difficulté on utilise soit RIA_VISER ou IA_VISER qui retourneront une coordonnées. On met à jour le tableau en utilisant la coordonnées, puis on utiliser la fonction toucherCouler pour vérifier si un bateau à coulé. On recommence ensuite cette opération si l'intelligence artificielle à réussi à toucher un bateau et si la partie n'est pas fini, et nous retournons enfin le tableau lorsque cette condition n'est plus vraie.

IV Difficultés rencontrées

Nous voulions avoir un programme complet avec un bon affichage et plusieurs possibilités. Ainsi ceci nous a valu deux week-ends bien chargés en codage mais les règles et conditions ont été respectés et nous en sommes heureux. Voici les plus grosses difficultés que nous avons rencontrées :

- La saisie des bateaux : ceci nous a demandé une grande réflexion pour obtenir une fonction optimisée puisque la restriction d'avoir au minimum un couloir d'eau entre chaque bateau et que tout le bateau soit dans le plateau de jeu amène vite à écrire énormément de lignes de codes.

- Compréhension de la bibliothèque ncurses : cette bibliothèque nous a été très utile puisqu'on y retrouve les couleurs que nous avons utilisées, la possibilité d'utiliser les flèches du clavier ou encore les constantes ACS_URCORNER, ACS_ULCORNER, ACS_LRCORNER et ACS_LLCORNER permettant d'obtenir un meilleur affichage des coins du tableau, ou même des constantes pour l'affichage du coeur du plateau, nous allons bien évidemment pas toutes les enumérer. La bibliothèque ncurses a donc une grande importance dans notre programme, il a donc fallu apprendre à l'utiliser ce qui a été compliqué et nous a demandé beaucoup de recherches.

- Une intelligence artificielle compétente : nous avons créé deux niveaux de difficulté pour notre intelligence artificielle, le niveau facile joue toujours aléatoirement alors que le niveau moins facile réfléchit à chaque tour à comment tirer au bon endroit. Les fonctions qui ont créé cette intelligence artificielle ont été décrite précédemment. Cette intelligence artificielle compétente nous a demandé beaucoup de temps et de réflexion puisque nous voulions que sa réflexion ressemble au maximum à la réflexion humaine.

V Contributions de chaque personne du groupe

Pour ce qui est de l'organisation au sein du groupe, on a décidé de se répartir les différentes tâches telle que la gestion de l'intelligence artificielle, l'affichage et le fonctionnement du jeu. Les fonctions n'ont pas un auteur en particulier puisque lorsqu'une fonction a été créée, les autres membres du groupe ont essayé de comprendre le raisonnement que le programmeur a eu. C'est ainsi que des idées d'optimisation nous sont venues, il fallait des fois revenir sur la structure même de la fonction alors que le raisonnement était quasi-similaire. Nous avons donc pris l'initiative de ne pas signer les fonctions que nous avons créée puisqu'elles étaient en réalité fabriquées à partir d'une idée de chacun. Le temps était assez court ce qui nous a contraint à faire les dernières fonctions chacun de notre coté.

VI Notre bilan personnel

Nicolas Dos Santos : Dans l'ensemble, mon bilan personnel est positif. En effet, grâce à mes deux camarades Théo JULIEN et Maxime PEGANE-VINGADAS j'ai eu l'occasion de progresser en codage tout en donnant aussi de mon expérience. Ce projet m'a permis aussi d'apprendre à utiliser une bibliothèque tel que ncurses ou encore à réfléchir à comment coder une intelligence artificielle et rassembler mes connaissances et celle de mes camarades afin d'obtenir le programme le plus optimisé possible. Le délai était très court mais je suis fière de notre programme malgré des améliorations que nous aurions pu ajouter, bien qu'un programme puisse toujours être amélioré.

Théo Julien : Pour ma part, ce projet m'a permis de découvrir différentes fonctions et utilisation du langages de programmation C. Le travail d'équipe et l'organisation avec mes caramades ont été crutial afin de finir la totalité du projet en temps voulu. Evidemment, la limite de temps nous a restreint aux fonctionnalitées demandées/proposée, si le temps nous l'aurait permis nous aurions pu améliorer de nombreux points. Cependant ce projet m'a été très instructif et bénéfique car il obligeait la coopération dans l'équipe et la recherche de documentation sur les fonctions que nous avons pu utiliser.

Maxime Pegane-Vingadas : Je pense que ce projet m'a aider à solidifier mes connaissance en C, que j'ai apprises tout au long de ce module. J'ai pu m'interesser au concept d'intelligence artificielle et me suis même essayé à la coder. Cependant je trouve que ce que l'on à créer mes camarades et moi se rapproche plus d'un arbre de possibilité un peu élaboré plutot qu'une réelle intelligence artificielle. Je pense aussi qu'avoir plus de temps aurait grandement bénéficié l'ensemble des groupes, que ce soit pour terminer plus proprement le programme ou ajouter d'autre options de gameplay et de personalisation. En somme ce projet m'a aider renforcer mes connaissance mais son délai est trop court.