Université Libre de Bruxelles
Faculté des Sciences
1ère candidature en Informatique

Algorithmique et Programmation : Projet 5
Combat Naval

Dans le cadre de ce projet, il vous est demandé de réaliser un jeu de Combat Naval.

Règles :

Chaque joueur (dont l'un des deux sera l'ordinateur) possède deux grilles de 10 cases par 10 cases.

Au début du jeu, chacun doit placer sur la première des deux grilles les cinq bateaux suivants :
- un porte-avions : 5 cases ;
- un cuirassé : 4 cases ;
- un croiseur : 3 cases ;
- un sous-marin : 3 cases ;
- un destroyer : 2 cases.

Il va sans dire que toutes les cases d'un bateau sont alignées et contiguës. Il faudra néanmoins respecter certaines règles pour le placement de la flotte :
- les navires ne peuvent toucher un côté de la grille que par une de leurs extrémités, ils ne peuvent donc pas *longer* un côté de la grille;
- les navires ne peuvent pas se toucher par les cotés, mais peuvent se toucher pas un coin.

Lors du jeu, chacun à son tour, un joueur annonce une case où il désire tirer, en donnant une lettre (de A à J), symbolisant le numéro de la ligne, et un nombre (de 1 à 10), symbolisa˝2000˝nt le numéro de la colonne. L'adversaire répond ensuite « Touché » ou « Dans l'eau », suivant que le tir ait atteint ou non une cible. Si le bateau atteint est entièrement détruit, la victime annonce « Coulé ». L'attaquant peut ensuite marquer sa deuxième grille en fonction de la réponse, ce qui lui permet d'orienter la suite de ses coups.
Le gagnant est le premier qui a touché les 17 cibles de son adversaire.

Votre travail :

On vous demande de réaliser une implémentation en C++ de ce jeu dans une version Humain contre Ordinateur.
Le jeu se déroulera en deux phases :
1ère phase : le joueur et l'ordinateur (voir plus loin) placent leurs bateaux sur leurs premières grilles respectives. La grille du joueur humain est rentrée par celui-ci (au clavier ou dans le programme), quant à l'ordinateur, il sélectionne une grille au hasard parmi une collection de grilles pré-établies, que nous vous remettrons (voir l'explication plus détaillée ci-dessous). L'ordinateur vérifie ensuite que la grille de l'utilisateur est correcte (en fonction des règles citées ci-dessus).
2ème phase : le jeu proprement dit ; alternativement, l'ordinateur (selon une stratégie définie ci-après) et le joueur proposent un coup et reçoivent la réponse de leur adversaire.

Le jeu doit se dérouler de la façon suivante :
Le programme demande au joueur de remplir sa grille (le libre choix de la méthode vous est donné).
Une fois vérifié que la grille est correcte, le jeu peut commencer (si elle n'est pas correcte, on le signale et on demande à l'utilisateur d'introduire une nouvelle grille, et ainsi de suite jusqu'à ce que la grille soit correcte).
L'utilisateur et l'ordinateur peuvent alors, chacun à son tour, introduire leur coup. L'affichage (les deux grilles, l'une avec bateaux du joueur et les coups de l'adversaire, l'autre avec les coups du joueur) peut alors être mis à jour.
On recommence ainsi, jusqu'à ce qu'un des deux adversaires ait coulé la totalité de la flotte adverse.

Le joueur ne doit pas lui-même répondre à l'ordinateur si les coups sont ratés ou ont atteint leurs buts, l'ordinateur s'en charge.

Stratégie de l'ordinateur :

L'ordinateur a principalement deux choses à faire : placer ses bateaux au départ et choisir intelligemment ses coups (d'après une certaine stratégie).

1.Placement des bateaux

Nous vous fournissons un fichier contenant la déclaration de 5 matrices (en fait un vecteur de 5 matrices). Il se trouve sur le réseau DI-NET à l'adresse « P:\cours\informattique\candi1 API\Projet5\grilles.h ». Vous devrez inclure ce fichier dans votre programme à l'aide d'une commande « #include "grilles.h" », après l'avoir copié dans le même répertoire que votre projet. Vous pourrez ensuite choisir aléatoirement (grâce à une fonction qui vous est donnée ci-dessous) une des 5 matrices et l'employer pour les emplacements des bateaux de l'ordinateur.
Le format de chacune de ces matrices est celui suggéré plus loin.

2. Stratégie de l'ordinateur

Il y a 3 étapes distinctes et successives dans la stratégie d'attaque :

1ère étape : "random", c'est à dire que l'ordinateur va choisir un coup au hasard sur la grille là où il n'y a pas encore eu de tir. Pour réaliser cela, il suffit de tirer une case au hasard (par la méthode décrite ci-dessous), et de vérifier qu'elle n'a pas déjà été bombardée. Si c'est le cas (déjà bombardée), il faut alors choisir une autre case et répéter cela jusqu'à ce que la case choisie soit vierge.
2ème étape : « recherche de la direction ». On se trouve dans cette phase lorsqu'un bateau a été touché pour la première fois. Il faut alors trouver dans quelle direction le bateau est placé. La technique consiste à bombarder les 4 cases autour de ce point jusqu'à trouver la suite du bateau. Données nécessaires : la position de la case « touchée ». Il faut bien sûr également veiller à marquer certaines cases comme étant « dans l'eau » (car, p.ex., les bateaux ne peuvent se toucher !).
3ème étape : « bombardement ». Un bateau pas encore coulé mais qui a au moins 2 cases touchées doit être bombardé sur la droite qui passe par ces 2 points jusqu'à ce qu'il soit coulé. Bien entendu, si un tir est « dans l'eau » mais que le bateau n'est pas encore coulé, Il faut continuer à le bombarder mais à l'autre extrémité.

Pendant le jeu, l'ordinateur se trouve toujours dans une de ces trois phases. En fait, il va commencer par la première, passer à la deuxième, puis à la troisième et ensuite revenir à la première. Il se peut cependant que l'on ne passe pas par la troisième phase dans le cas du destroyer (2 cases).

Remarques :

Les données nécessaires décrites sont une indication, il se peut que des données supplémentaires soient utiles.

Lorsqu'un bateau est coulé, on est sûr que les cases directement au-dessus, en dessous, à gauche et à droite sont de l'eau. Elles sont alors considérées comme bombardées et « Dans l'eau ».

Suggestions :

- Pour la matrice où les bateaux sont placés, nous vous conseillons d'utiliser une matrice où seront placés un '1' dans les cases occupées par le premier bateau (le porte-avions, 5 cases), un '2' pour le deuxième (le cuirassé), etc. (3ème=croiseur, 4ème=sous-marin, 5ème=destroyer). Les autre˝2000˝s cases (l'eau), seront remplies de '0'. Cela vous facilitera grandement la vérification du départ.
- Nous vous suggérons d'utiliser la découpe en classe suivante:
  - Une classe Jeu, jouant principalement le rôle d' « arbitre »,
  - Une classe JoueurHumain contenant (entre-autre) les fonctions suivantes :
    Tirer(), renvoyant le coup tiré par l'utilisateur;

    Recevoir_Tir(point) : cette fonction reçoit les coordonnées du coup tiré par l’ordinateur (dans la variable « point ») et renvoie le résultat du coup (ex : 0 : touché, 1 : coulé, 2 : dans l’eau).
    Donner_Resultat(resultat), indiquant au joueur le résultat du dernier tir (càd le résultat du tir que le joueur a opéré par la fonction Tirer())

- o Une classe JoueurVirtuel, contenant les mêmes fonctions, mais avec, bien entendu, des contenus pouvant être différents. Par exemple, la fonction Tirer() de l’ordinateur emploie un choix aléatoire (voir ci-dessous), alors que celle du joueur saisit au clavier le choix de celui-c˝2000˝i.

La classe JoueurHumain gèrerait également l’affichage de ses deux grilles (les grilles de JoueurVirtuel ne doivent bien évidemment pas être affichées).
Le rôle de la classe Jeu se résumerait alors, dans la phase de jeu, quand A attaque B, à exécuter une suite d’appels de fonctions, dont une partie serait du genre :

```
point =A.Tirer() ;
resultat = B.Recevoir_Tir(point) ;
A.Donner_Resultat(resultat) ;
```

N’oubliez qu’il faut pouvoir déterminer que le jeu est fini.

Il va sans dire que l’objet JoueurHumain connaît les grilles du joueur, que l’objet JoueurVirtuel connaît les grilles de l’ordinateur, mais qu’aucun des deux objets ne connaît les grilles de l’autre objet.

Remarques :

- Pour générer un nombre (pseudo)aléatoire entre 0 et n-1 (bornes comprises), il vous faut :
    - o inclure les deux librairies standard stdlib.h et time.h en début de fichier (avec #include <stdlib.h> et #include <time.h>),
    - o initialiser le générateur en appelant la fonction srand(time(0)) en début de programme,
    - o vous pouvez maintenant générer votre nombre (entier) en appelant
      r = int(double(rand())/R [1]; AND_MAX*n)
- L’aspect graphique de votre travail ne sera pas pris en compte. Une interface en mode texte est largement suffisante.
- Nous insistons particulièrement sur la lisibilité de votre code ; noms de variables et de fonctions judicieusement choisis, indentation correcte, commentaires pertinents, …

Le projet (listing et rapport) est à remettre pour le vendredi 7 décembre aux assistants de vos groupes de TP. Pour des raisons de test, le code source et l’exécutable devront être sur votre compte sur DI-NET, dans un répertoire d:\Projet5. Nous désirons que le programme soit implémenté dans un seul fichier « .cpp », éventuellement complété par des fichiers « .h ». Votre projet devra être compilable avec le compilateur Borland disponible aux salles du NO.
N’oubliez par d’indiquer sur chaque page de votre projet votre nom, ainsi que votre groupe et, au moins sur la première page, votre login.

Bon travail !

---

[1] Si vous désirez obtenir la même suite de nombres (pseudo)aléatoires lors de 2 exécutions différentes du programme, il suffit de ne pas appeler la fonction « srand », et d’employer directement la fonction « rand » comme montré dans le texte.