IFB : Rapport de projet
Bataille navale

ESCANDE Anaïs
TC02 UV-IFB

BONNEFOY Aubin
TC02 UV-IFB

Groupe E

Table des matières

Introduction

Dans le cadre de l'unité de valeur IFB, dispensée en TC02, nous sommes amenés à réaliser un projet en langage de programmation C. Ce projet a pour but de nous évaluer sur les bases de programmation en C, vues au cours de ce semestre. Parmi les deux sujets proposés, nous avons choisi de coder une bataille navale.

L'objectif du jeu est d'anéantir la flotte navale gérée par l'ordinateur. Pour cela, le joueur disposera de plusieurs niveaux de difficulté (facile, moyen, difficile), de plusieurs modes de jeu (classique, blind, active) et de plusieurs types de missiles (simple, tactique, artillerie, bombe). L'utilisateur pourra choisir à la fin de chaque tir de continuer à jouer ou sauvegarder et quitter. Au début de chaque partie il choisira de reprendre une ancienne partie ou d'en commencer une nouvelle.

Ainsi, nous allons présenter notre organisation, la structure générale du code, les difficultés rencontrées, puis une analyse du code et les points à améliorer.

Organisation

Nous avons commencé par étudier les deux sujets proposés, afin de déterminer lequel nous semblait plus intéressant et les points qui pourraient nous poser problèmes sur chaque projet. Nous avons choisi la bataille navale pour la combinaison de stratégie et hasard au sein d'un même jeu. Une fois cette première étape réalisée, nous avons regardé le code dans son ensemble pour réaliser un algorithme général et déterminer les grands axes du programme.

Puis, nous avons fait le choix de fixer des délais pour chaque mode à réaliser, que nous adaptions chaque semaine en fonction de l'avancement. Nous nous retrouvions une fois par semaine pour mettre en commun ce qui avait été fait et pour étudier ensemble les problèmes rencontrés afin de chercher des solutions et préciser certains algorithmes. Notre répartition des tâches se faisait de semaine en semaine, et nous pouvions suivre nos avancements mutuels en utilisant GitHub. On avait ainsi une vision de ce qu'il restait à faire par mode et nous nous répartitions les tâches en fonction des parties qu'on visualisait mieux respectivement. Aubin étant plus à l'aise avec ce langage, il codait donc les parties plus complexes, et Anaïs réalisait certains algorithmes et les fonctions plus simples, ou les bases de celles-ci.

Nous avons cependant mal pensé notre algorithme global et aurions pu nous attarder davantage sur celui-ci. En effet nous avons rédigé celui-ci étape par étape, en fonction des différents modes du cahier des charges. L'algorithme et les fonctions réalisées n'étaient donc pas compatibles avec tous les modes, ce qui nous a contraints à plusieurs reprises de repenser des fonctions existantes pour les adapter à chaque subtilité apportée par les autres règles. Notre temps n'a donc pas été optimisé et cela a entrainé un certain retard.

Structure du code

Le programme comporte plusieurs modules permettant de structurer le code et ainsi de le rendre plus compréhensible.

Initialisation grille

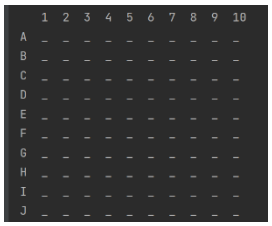
Ce module permet de générer la partie en plaçant les bateaux sur la grille et en affichant la grille de jeu à l'utilisateur. Voici les différentes fonctions qui le permettent :

Init_grille

Permet de créer la grille de jeu de taille 10 par 10 et affecte le caractère « _ » aux cases vides.

Show_grid

Elle affiche la grille de jeu à l'utilisateur et les grilles de bateaux, pour faire des vérifications de code et contrôler l'avancement d'une partie (ces contrôles sont disponibles sur des lignes commentées par « //vérif code ». Elle affiche les colonnes de 1 à 10 et les lignes de A à J.



Une fois la grille initialisée dans la fonction principale, on appellera les fonctions suivantes pour chacun des cinq bateaux :

Genere_bateaux

La fonction donne pour chaque bateau une orientation verticale ou horizontale, ainsi qu'une coordonnée en x et une coordonnée en y.

On effectuera les tests suivants tant que toutes les conditions ne seront pas satisfaites :

App_grille

Grâce à cette fonction, on vérifie que les bateaux générés soient contenus dans la grille.

Chevauchement

Cette fonction contrôle s'il n'y a pas de bateau dans la ligne où le bateau est généré dans le cas d'un bateau horizontal, et dans la colonne pour un bateau vertical.

Après que les bateaux aient été correctement placés, on appelle la dernière fonction :

Placement_bateaux

Une fois les coordonnées et l'orientation déterminés, la fonction détermine la position totale du bateau grâce à sa taille. Pour une orientation verticale, le point de coordonnées (CooX, CooY) est le point le plus haut du bateau, pour une orientation horizontale c'est le point le plus à gauche du bateau.

Exécution_tir

Ce module contient tous les éléments permettant au joueur de tirer sur les bateaux.

Choix_coo_de_tir

Cette fonction demande à l'utilisateur et récupère, la ligne et la colonne dans laquelle il veut tirer. Elle affiche un message d'erreur et réitère l'acquisition si les coordonnées saisies ne sont pas correctes.

```
Dans quelle colonne souhaitez vous tirer :5
Dans quelle ligne souhaitez vous tirer :k
Veuillez saisir une lettre entre A et
J Dans quelle ligne souhaitez vous tirer :e
```

Choix_missiles

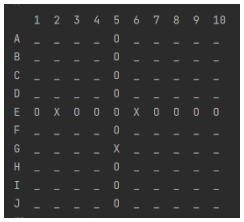
La fonction demande à l'utilisateur quel missile il souhaite tirer. Elle comprend un contrôle d'acquisition et affiche un message d'erreur et demande à l'utilisateur de saisir à nouveau le missile choisi.

```
Veuillez saisir un des missiles suivants :
 -A : Pour missile d'artillerie
 -T : Pour missile tactique
 -B : Pour bombe
 -S : Pour missile simple
A
```

```
ERREUR, vous n'avez plus de ce missile
Veuillez saisir un des missiles suivants :
 -A : Pour missile d'artillerie
 -T : Pour missile tactique
 -B : Pour bombe
 -S : Pour missile simple
```

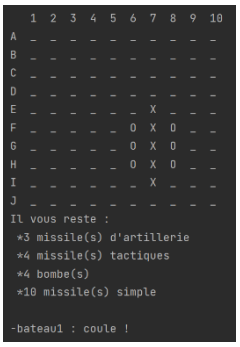
Fire_artillery

Le missile d'artillerie touche toutes les cases se trouvant sur la même ligne ou la même colonne que le point d'impact. La fonction affiche « X » pour les cases où se trouve un bateau et « O » pour les cases ou le missile tombe dans l'eau.



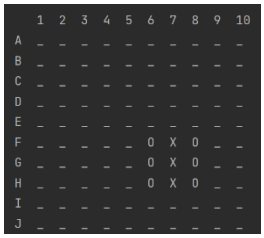
Fire_tactical

Le missile tactique lorsqu'il touche un navire, le coule immédiatement même s'il restait des cases non touchées sur le navire. La fonction affiche alors des « X » sur la totalité du bateau. S'il n'y a pas de bateau, la fonction affiche un « O » sur le point d'impact.



Fire_bomb

La fonction permet que toutes les cases dans un certain rayon autour du point d'impact soient touchées. Pour les cases où se trouve un bateau sont marquées par un « X » et les autres non touchées par un « O ».



Fire_simple

Le missile simple touche la seule case ciblée. Si le point d'impact touche un bateau, la fonction affiche un « X », sinon un « O ».

Void_tir

La fonction déclenche le missile sélectionné et retire 1 missile à l'inventaire. S'il n'y a plus du type de missile sélectionné, la fonction affiche un message d'erreur.

État-partie

Bateaux_restants

Cette fonction permet de déterminer et afficher pour chaque bateaux le nombre de cases restantes pour couler chaque bateau, ainsi que le nombre de bateau qu'il reste à couler.

```
-bateau1 : 4 cases restantes
-bateau2 : 4 cases restantes
-bateau3 : 2 cases restantes
-bateau4 : 2 cases restantes
-bateau5 : 2 cases restantes

Il vous reste 5 bateaux a couler
```

Missilles_restants

Cette fonction permet uniquement d'afficher le nombre de missiles restants pour chaque catégorie de missiles.

```
Il vous reste :
 *9 missile(s) d'artillerie
 *10 missile(s) tactiques
 *10 bombe(s)
 *10 missile(s) simple
```

Check_loose

Vérifie le nombre de missiles du joueur. S'il n'y a plus de missiles toutes catégories confondues elle affiche au joueur que la partie est finit.

```
Vous n'avez plus de missiles. Vous avez perdu !

Process finished with exit code 0
```

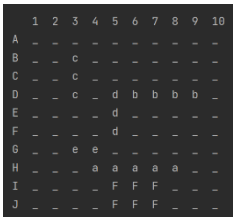
Les fonctions suivantes s'appliquent au mode Active :

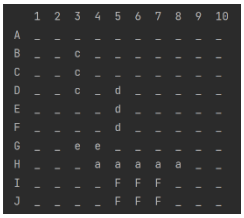
Bateau_à_déplacer

Cette fonction vérifie pour chaque bateau s'il a été touché, puis génère aléatoirement un nombre entre 1 et 5 et vérifie si le bateau correspondant peut être déplacé. Si aucun bateau ne peut être déplacé la fonction le notifie à l'utilisateur.

Sup_ancienne_position

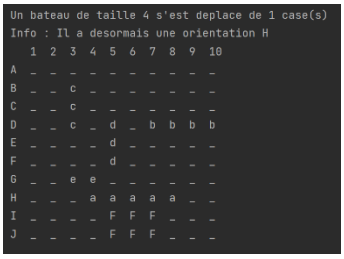
Parcours le bateau à déplacer et remplace dans la grille qui n'est pas visible par l'utilisateur par des cases vides. La position du bateau a été supprimée. (Ici le bateau b).





Nouvelle_position

Cette fonction attribue une nouvelle position au bateau à déplacer. Elle affecte une nouvelle orientation au bateau, détermine de combien de cases et dans quel sens le bateau sera déplacé et répète cette opération tant qu'il sera hors de la grille ou sur la position d'un bateau existant. Elle indique les changements à l'utilisateur, puis replace le bateau sur la grille qui n'est pas visible par l'utilisateur.



Affichage_cases_blind

Cette fonction parcourt la grille dédiée au mode blind qui n'est pas visible par l'utilisateur, et notifie au joueur le(s) point(s) d'impact sur un ou plusieurs bateaux.

```
-Vous avez touche un bateau en (4,F)
-Vous avez touche un bateau en (5,F)
-Vous avez touche un bateau en (6,F)
-Vous avez touche un bateau en (6,H)
```

Type_de_partie

Classique

Cette fonction représente le mode de jeu classique. Elle appelle les fonctions init_grille, check_loose, choix_coo_de_tir. Si la case sélectionnée a déjà été touchée, la fonction affiche un message d'erreur et redemande la saisie à l'utilisateur. Elle appelle les fonctions choix_missile, et show_grid. Puis elle affiche le nombre de missiles et de bateaux restants avec les fonctions missiles_restants, bateaux_restants. Enfin elle demande à l'utilisateur s'il souhaite continuer à jouer, et affiche un message d'erreur en cas de saisie incorrecte.

Blind

La fonction Blind possède la même structure que la fonction classique à l'exception qu'elle n'affiche pas la grille de jeu lors de la partie du joueur (la fonction show_grid n'est pas appelée).

Active

Dans ce mode la grille de jeu est affichée. A chaque tour, un des bateaux change de position, son orientation peut changer et il peut se déplacer d'une à trois cases. Seuls les bateaux qui n'ont pas encore été touchés pourront être déplacés, à condition qu'ils ne chevauchent aucun autre bateau. Le joueur ne bénéficiera donc pas des indications de « tir dans l'eau » dans la grille de jeu, un bateau étant susceptible de s'y retrouver au tour suivant. La structure sera la même que le mode classique et à la fin de chaque tour la fonction appellera

les fonctions : bateau_a_deplacer, supp_ancienne_position, nouvelle_position, qui affecteront une nouvelle position à l'un des bateaux.

Lancement_nouvelle_partie

Si le joueur souhaite démarrer une nouvelle partie, la fonction affiche les différents modes, récupère le choix de l'utilisateur et lance la partie. En fonction de la réponse de l'utilisateur elle appellera la fonction classique, blind ou active.

```
Veuillez saisir le mode de jeu souhaite :
 -C : Mode Classique
 -B : Mode Blind
 -A : Mode Active
c
```

Choix_difficult

Le jeu dispose de différents niveaux de difficultés, la fonction propose ainsi le niveau facile, moyen, et difficile. Elle récupère le choix de l'utilisateur, avec message d'erreurs, tant que la saisie sera incorrecte. Une fois le choix de l'utilisateur terminé, la fonction initialisera le nombre de missiles selon la difficulté sélectionnée.

```
Quel mode souhaitez-vous ? (Facile/Moyen/Difficile)
facile
Vous avez :
 10 missile(s) d'artillerie
 10 missile(s) tactiques
 10 bombe(s)
 10 missile(s) simple
```

Sauvegarde

Save

Cette fonction permet de sauvegarder le mode de jeu, le nombre de missiles restants ainsi que la grille de bateau d'une partie en cours en recopiant celle-ci dans un fichier.txt.

Load

Cette fonction permet de relancer une ancienne partie dans laquelle le joueur a déjà effectué des tirs de missile. Lorsque le joueur choisit de charger une ancienne partie, la fonction ouvre le fichier.txt créer avec la fonction save, puis elle affecte la valeur du tableau se trouvant de ce fichier.txt au nouveau tableau de bateau afin de l'initialiser. Cette fonction se charge aussi d'affecter le nombre de missiles restants ainsi que d'ouvrir le bon mode de jeu (Classique/Blind/Active).

Init_save

Puisque le programme ne sauvegarde que la grille contenant les bateaux, il est nécessaire de réinitialiser la grille de jeu. Pour cela nous utilisons cette fonction init_save qui prend en paramètre la grille de bateau récupérée grâce à la fonction load afin d'attribuer 'X' aux bateaux cases non vides touchées par un missile, '0' aux cases vides touchées par un missile et '_' aux cases non touchées durant la partie sauvegardée.

```
Que choisissez vous ?
 - C : Charger une ancienne partie
 - N : Nouvelle partie
 - Q : Quitter
n
```

Les problèmes rencontrés

La fonction chevauchement

Pour commencer, nous avons rencontré un problème avec les fonctions destinées à faire en sorte que les bateaux générés soient correctement placés dans la grille et notamment qu'ils ne se chevauchent pas.

La fonction chevauchement consiste simplement à parcourir le dernier bateau généré sur toute sa longueur, afin de vérifier que les cases qui lui sont associées sur la grille de bateaux ne soient pas déjà occupées par un autre bateau généré auparavant (évitant ainsi un éventuel chevauchement). Voici une illustration de cette fonction lorsqu'elle était erronée :

```
if(bateau[i].orientation == 'H'){
    for(a = bateau[i].position_x; a - bateau[i].position_x < bateau[i].taille; a++){
        if(grille_bateau.grille[a][bateau[i].position_y] != '_'){
            test = 1;
        }else{
            test = 0;
        }
    }
}else{
    for(a = bateau[i].position_y; a - bateau[i].position_y < bateau[i].taille; a++){
        if(grille_bateau.grille[bateau[i].position_x][a] != '_'){
            test = 1;
        }else{
            test = 0;
        }
    }
}

return test;
```

Comme nous pouvons le voir le fonctionnement de cette fonction est relativement simple, en effet, le principe étant de vérifier pour une case du bateau si elle est déjà occupée ou non sur la grille de bateau, si c'est le cas la fonction retourne '0' et dans le cas contraire '1', puis on passe à la case suivante et on recommence jusqu'à la fin du bateau en question.

Cette fonction agit donc comme un testeur qui va indiquer '0' au programme si le bateau qu'il vient de générer peut-être positionner dans la grille sans chevaucher d'autres bateaux et '1' sinon. Or cette partie de code ne fonctionne pas pour une raison très simple c'est que la variable 'test' qui lorsqu'une case est déjà attribuée à un autre bateau sur la grille prend la valeur '1' devrait tous simplement être initialiser à '0' et lorsqu'elle passe à '1', ne plus jamais pouvoir revenir à '0'. Donc le else est en trop dans cette fonction car avec le else cette fonction nous indique seulement si la dernière case du bateau est libre ou occupée.

Ainsi, le problème que nous a posé cette fonction résulte essentiellement du fait que nous n'avons pas pris assez de recul, car par une simple analyse de celle-ci comme nous venons de le faire, il est très simple de le résoudre.

Le mode Blind

Le second problème que nous avons rencontré est relié aux modes de jeu (Facile/Classique/Blind) et principalement pour le mode Blind que nous avons réalisé après le mode Classique. Pour rappel, la différence entre ces deux modes est que le mode Classique affiche la grille de jeu avec les cases où des tirs ont été effectués, on peut ainsi y observer des 'X' pour des bateaux touchés 'O' lorsque le missile est tombé dans l'eau et '_' lorsqu'aucun missile n'a été tiré sur cette case. Le mode Blind lui ne doit pas afficher la grille de jeu entre chaque tour, mais doit indiquer les cases qui ont été touchées uniquement pour ce tour.

Dans un premier temps nous avons pensé à indiquer quelles cases étaient touchées entre chaque tour en mode Classique, ainsi cela nous aurait permis de simplement réutiliser la fonction de ce mode en mode Blind en supprimant simplement la fonction 'show_grid' qui affiche la grille entre chaque tour. Or nous nous sommes dit que cela était un peu trop facile et ne répondait pas forcément à la consigne. Le problème qui se posait était donc de trouver un moyen d'indiquer quelles cases avaient été touchées lors du dernier tir.

Nous avons donc eu pour projet de lire les cases touchées sur la grille de jeu à l'aide d'une fonction or cela entrainait un autre problème. En faisant cela, nous lisions toutes les cases touchées depuis le début de la partie, or la consigne demandait d'indiquer les cases touchées durant le dernier tour.

Nous avons donc longtemps réfléchi à une manière pour résoudre ce problème tout en trouvant une solution la plus optimale possible. Notre première idée était d'effectuer un test directement dans les fonctions missiles qui relèverait les cases touchées lors de l'exécution du tir et les stockeraient dans un tableau pour ensuite les lire, or cela nous obligeait à faire d'énormes modifications sur des fonctions parfaitement adaptées à notre code et nous n'étions pas sûrs que cette solution soit très optimale.

Finalement, nous avons opté pour une solution, à notre avis, bien plus efficace. Celle-ci consiste en la création d'une troisième grille que l'on a appelée 'cases_touchées' (pour rappel nous avions déjà deux grilles : une destinée à être affichée au joueur et l'autre

contenant les bateaux) sur laquelle nous affichons uniquement les points d'impacts ('X') entre un missile et un bateau. Cette grille est réinitialisée entre chaque tour grâce à la fonction 'init_grille'. Ainsi pour indiquer au joueur qu'elles cases il a touché au cours de son dernier tir, il suffit de parcourir la grille 'cases touchées' à la fin du tour et de lire les coordonnées des point d'impact ('X').

Pour résoudre ce problème il nous suffisait juste d'envisager une autre méthode et ne pas rester bloqués sur notre première idée bien plus difficile à réaliser.

Le mode active

Nous avons aussi rencontré quelques problèmes lors de la création du mode active (pour rappel en mode active un bateau non touché peut se déplacer aléatoirement de 1 à 3 cases entre chaque tour. Pour cela on supprime l'ancienne position d'un bateau sur la grille, puis on lui en attribue une nouvelle). L'attribution de cette nouvelle position est assurée par la fonction 'nouvelle_position' et c'est celle-ci qui nous a posée des problèmes.

Pour déplacer un bateau d'une à trois cases, la fonction 'nouvelle_position' prend tous simplement en paramètre les anciennes positions du bateau ainsi que son orientation (par le biais de la structure Boat). On génère une nouvelle orientation aléatoirement à l'aide d'un rand, puis, si cette orientation est horizontale on ajoute un nombre compris entre -3 et 3 à l'ancienne position en abscisse du bateau. Inversement si l'orientation générée est verticale ce nombre compris entre -3 et 3 est ajouté à l'ancienne position en ordonnée. Pour finir afin de s'assurer que le bateau ne dépasse pas de la grille et ne chevauche pas un bateau déjà placé sur la grille on fait appel aux fonctions 'chevauchement' et 'app_grille' et on recommence l'opération tant que ces fonctions retournent une erreur.

Le problème que nous avons rencontré est que la fonction ne parvient pas toujours à trouver une nouvelle position qui convient pour le bateau et dans ce cas-là le code s'arrête. Malgré nos efforts, nous avons réussi à faire en sorte que le bateau obtienne une nouvelle position dans la majorité des cas, mais il arrive encore quelque fois que la fonction n'arrive pas à attribuer une nouvelle position.

Nous avons aussi un autre problème pour le mode active qui, cette fois-ci, concerne la sauvegarde. En effet, nous ne parvenons pas à lancer correctement une partie sauvegardée en mode active. Lorsque l'on souhaite charger une partie en mode active celle-ci démarre correctement, le joueur peut jouer le premier tour mais au moment du déplacement aléatoire d'un des bateaux, la partie s'arrête avec un message d'erreur de manière systématique.

Nous pensons que ces problèmes persistent à cause d'un mauvais enchainement des différentes fonctions dans ce mode, pour les éviter il aurait fallu organiser la totalité du programme sous forme de pseudo code avant de commencer à coder les fonctions en C. Cela aurait assuré une meilleure cohérence entre les fonctions et aurait certainement permis d'éviter de telles erreurs.

Possibilités d'améliorations

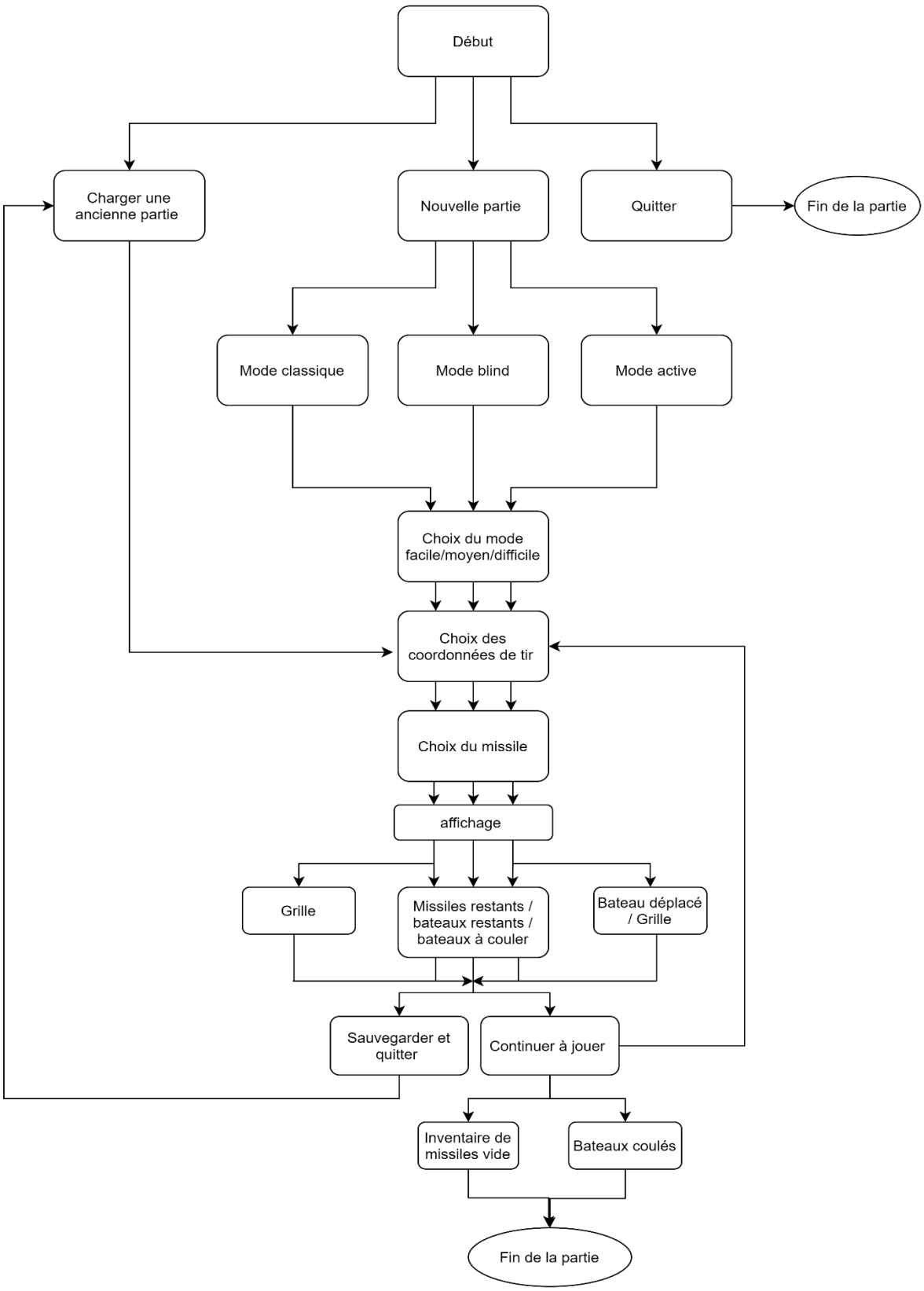
Nous pensons que notre projet pourrait être amélioré sur plusieurs points :

- Une optimisation du code est possible, en effet nous pensons que notre première erreur est de ne pas avoir envisagé le code dans sa globalité. Nous avons produit des algorithmes en pseudo-code pour chaque fonction et ensuite nous avons codé ses fonctions une par une en C. Néanmoins nous aurions dû faire un algorithme complet du programme en pseudo-code afin d'en envisager l'entièreté et ainsi d'avoir des fonctions cohérentes et réutilisables les unes avec les autres. Cela aurait permis d'offrir un code plus optimisé, plus ergonomique et beaucoup plus facile à comprendre par un correcteur.

- Nous aurions pu essayer d'enrichir notre structure 'Boat' qui pour rappel contient la taille, l'orientation, la position en abscisse et la position en ordonnée de chaque bateau sur la grille. En effet, les bateaux qui sont placés sur la grille dépendent de celle-ci, c'est-à-dire que nous les générons directement sur la grille et que nous vérifions leur caractéristiques (nombre de cases touchées par des missiles) en passant par la grille sur laquelle ils sont placés. Or nous pensons que si nous nous étions plus intéressés à cette structure 'Boat' nous aurions pu faire en sorte que chacun des bateaux soit totalement indépendants de la grille. Par exemple en ajoutant une variable dans la structure permettant de compter le nombre de cases touchées et non touchées pour un bateau (comme des points de vies). Nous pensons qu'en rendant les bateaux indépendants de la grille nous aurions pu rendre notre code beaucoup plus optimal et notamment simplifier certains points comme les sauvegardes.

- Le dernier point d'amélioration auquel nous pourrions nous intéresser si nous recommençons ce code, est l'utilisation de l'allocation dynamique de mémoire qui aurait pu nous être utile notamment avec les nombreux tableaux utilisés dans le programme. Cela aurait permis de ne pas utiliser plus de mémoire du système que nécessaire et ainsi de le rendre encore plus optimale. La principale raison, pour ne pas avoir fait appel à l'allocation dynamique de mémoire au sein de notre programme, est que son utilisation nous paraissait encore floue et malgré des tentatives d'une meilleure compréhension à l'aide d'internet, nous ne nous sentions pas capables de l'utiliser pour l'instant.

Annexe : Déroulement d'une partie



Conclusion

Pour conclure, ce projet d'informatique nous a permis d'utiliser les outils de cours de manière concrète et ainsi nous a donné une idée plus précise des difficultés qui existent dans la programmation. Nous avons par exemple pu nous rendre compte de l'importance de la conception d'un algorithme. La réalisation d'un jeu en entier a donc été l'occasion de se rendre compte de la nécessité d'un travail structuré et d'une bonne gestion du temps. Nous avons pu mettre à profit les qualités de chacun et en tirer profit pour l'aboutissement de celui-ci.

Remerciements

Nous tenons à remercier nos professeurs Monsieur Lombard ainsi que Monsieur Dubaj qui nous ont été d'une grande aide dans la résolution des problèmes rencontrés, malgré le peu de contacts que nous ayons eu en présentiel. Ils ont été disponibles et réactifs, nous leur en sommes très reconnaissants.