Bataille navale

1 Objectifs

1.1 Règle du jeu

Vous pouvez trouver une présentation de (variantes de) la bataille navale par exemple à l'adresse : `https://fr.wikipedia.org/wiki/Bataille_navale_(jeu)`

1.2 Objectifs généraux

Dans un premier temps (cf. sections 2 et 3), l'objectif est de réaliser une version de base de la bataille navale, dans laquelle le jeu se déroule de manière automatique. Les deux joueurs sont simulés lors de l'exécution du jeu, et les choix des configurations initiales et des tirs sont aléatoires.

La conception générale pour la réalisation de cette première version est imposée : il vous est demandé de concevoir, de développer et de tester les fonctions nécessaires.

Dans un second temps (cf. section 4), vous choisirez de nouvelles fonctionnalités à ajouter à la version de base. Il vous est demandé de concevoir et de spécifier ces fonctionnalités, et de concevoir, développer et tester les fonctions permettant de les réaliser.

Vous réaliserez ce jeu en binômes, en vous répartissant le travail de manière à ce que chaque personne ait une connaissance complète des réalisations. S'il est demandé d'écrire une fonction et des fonctions de tests, l'une des personnes écrit la fonction et l'autre écrit les fonctions de test : ces rôles doivent alterner, de manière à ce que chaque personne puisse écrire (à peu près) autant de fonctions que de tests.

1.3 Fichiers mis à disposition sur `UPdago`

Suivant que vous disposiez ou non de la bibliothèque `graphics`, vous récupérerez sur `UPdago` les versions adéquates des fichiers `CPbattleship.ml` et `CPbattleship_display.ml`.

Le fichier `CPbattleship.ml` contient la définition des types utilisés pour le développement de la version de base du jeu (cf. section 1.4 suivante) : vous compléterez le fichier en y ajoutant les fonctions que vous développerez. Ce fichier contient aussi la fonction `play`, qui est la fonction principale du jeu, mise en commentaire tant que les autres fonctions ne sont pas développées.

Le fichier `CPbattleship_display.ml` contient les fonctions appelées par la fonction `play` (ainsi que la définition des types nécessaires pour la version graphique), qui permettent d'afficher l'état des grilles des joueurs à chaque tour de jeu (cf. figure 1 pour l'affichage graphique). Le contenu de ce fichier doit être compilé avant de compiler la fonction principale `play`.

1.4 Types

Les types suivants sont définis dans le fichier `CPbattleship.ml` :

```
type t_value = EMPTY | SHIP | DAMAGED | TRIED ;;
type t_matrix = t_value matrix ;;
type t_position = {cx : int ; cy : int} ;;
type t_pos_list = t_position list ;;
type t_remaining_pos = t_pos_list ref ;;
type t_player = {name : string ; human : bool ;
                 mymat : t_matrix ; remain : t_remaining_pos ;
                 opp_mat : t_matrix ; opp_nb : int ref} ;;
type t_play = {pl1 : t_player ; pl2 : t_player} ;;
```

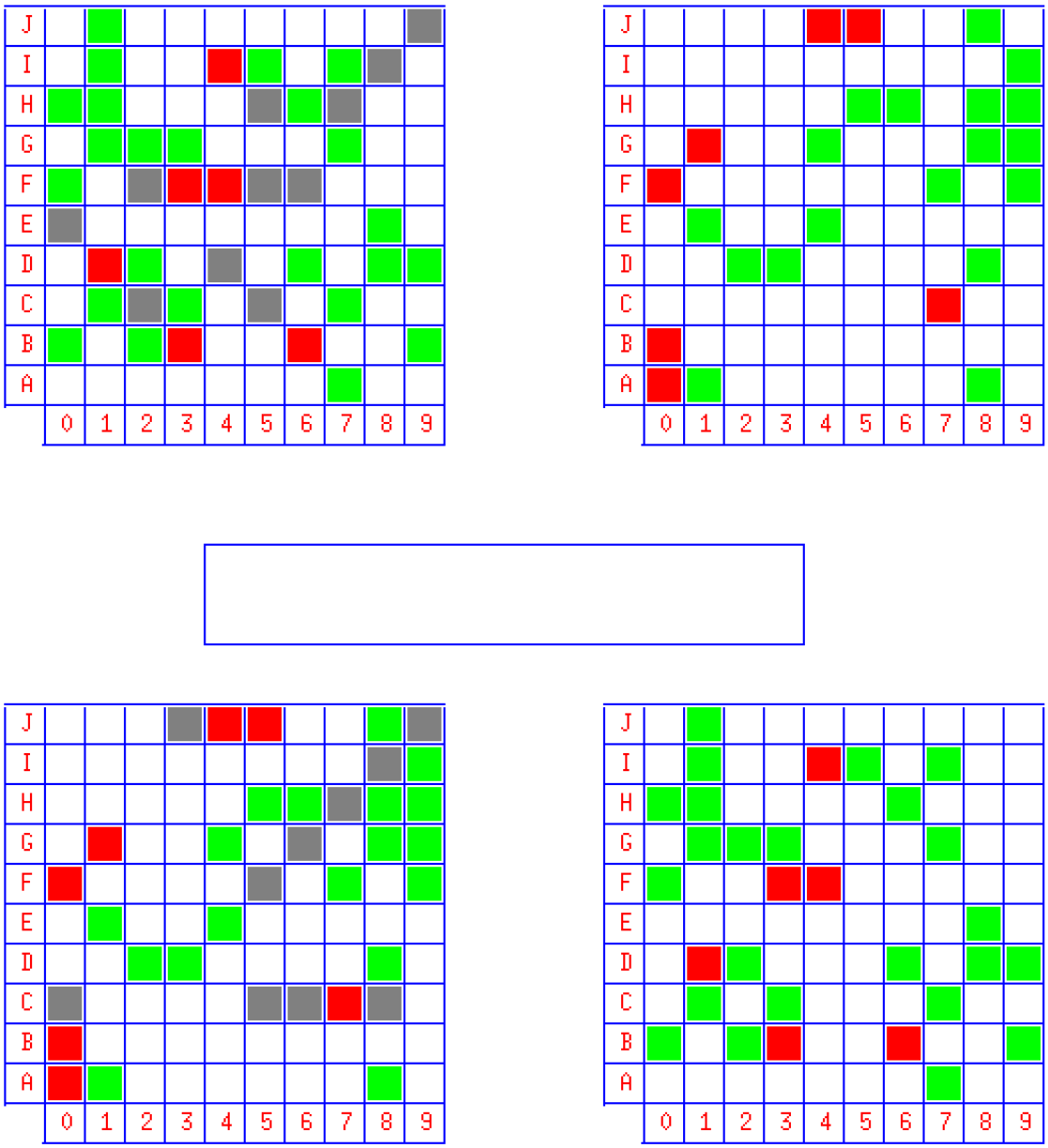

FIGURE 1 – Affichage graphique. En bas à gauche, la grille du premier joueur : les cases grises correspondent aux parties de ses bateaux non encore touchées ; les cases rouges correspondent aux parties touchées de ses bateaux. Les cases vertes correspondent aux tirs du deuxième joueur qui sont tombés à l'eau. En bas à droite, la grille indiquant ce que le premier joueur connaît de la grille du deuxième joueur (le code des couleurs est le même pour toutes les grilles) ; le joueur 1 sait qu'il a touché 6 cases correspondant à des parties des bateaux du joueur 2, et que ses autres tirs ont échoué. En haut, les grilles du joueur 2 : la grille propre du joueur 2 est à gauche, la grille correspondant à ce que le joueur 2 connait de la grille du joueur 1 est à droite. Bien sûr, les cases vertes et rouges de la grille en bas à droite sont identiques à celles de la grille en haut à gauche (de même pour la grille en haut à droite et celle en bas à gauche).

Le type `t_matrix` permet de représenter les grilles des joueurs ; le type `t_value` décrit les quatre valeurs possibles pour une case d'une matrice (appelée « grille » par la suite), c'est-à-dire `EMPTY` si la case de la grille est vide et n'a jamais été la cible d'un tir (en blanc sur la figure 1), `SHIP` si la case contient une partie intacte d'un bateau (en gris sur la figure 1), `DAMAGED` si la case contient une partie d'un bateau qui a été touchée par un tir (en rouge sur la figure 1), `TRIED` si la case est vide et a été la cible d'un tir (en vert sur la figure 1).

Le type `t_position` permet de représenter les coordonnées d'une case d'une grille. Le type `t_pos_list` permet de représenter des listes de coordonnées, et le type `t_remaining_pos` permet de stocker une liste de coordonnées dans une variable ou un champ mutable.

Le type `t_player` permet de représenter des joueurs. Le descriptif de chaque joueur contient les informations suivantes :

1. le nom du joueur (champ `name`) ;
2. le fait que ce soit un joueur humain ou simulé (champ `human`, qui a pour valeur `true` si le joueur est humain, `false` s'il est simulé) ;
3. la grille propre du joueur (champ `mymat`) ;
4. la liste des coordonnées des cases qui correspondent à des parties des bateaux du joueur qui n'ont pas encore été touchées par les tirs du joueur adverse (champ mutable `remain`). Cette information est redondante, puisque cette information peut être extraite de la matrice `mymat`. En l'état, cette information est inutile, mais elle sert de base pour une extension ;
5. ce que le joueur connaît de la grille du joueur adverse (champ `opp_mat`) ;
6. le nombre de parties de bateaux du joueur adverse qui n'ont pas encore été touchées (champ mutable `opp_nb`). Cette information aussi est redondante, puisqu'elle peut être calculée à partir du nombre total de parties de bateaux et de la grille `opp_mat` ; ceci dit, il est plus efficace (en temps de calcul) de gérer directement cette information, plutôt que de la recalculer.

1.5 Fonction `play`

Avant d'aller plus loin, prenez le temps d'analyser la fonction `play` afin de comprendre ce qu'elle fait. Elle prend en paramètre deux chaînes de caractères `nm_1` et `nm_2`, qui correspondent aux noms des deux joueurs, et deux booléens `hum_1` et `hum_2` qui indiquent si les joueurs correspondants sont humains ou s'ils doivent être simulés. L'affichage des matrices correspondant aux grilles des joueurs est effectué à l'initialisation et à chaque tour de jeu (le joueur 1 est en bas de l'affichage, le joueur 2 est en haut).

La variable `who` contient le numéro du joueur dont c'est le tour de jeu. La fonction `play_one_player`, en plus de réaliser un tour de jeu, a pour résultat le nombre de tirs erronés émis par le joueur dont c'est le tour de jeu (c'est-à-dire des tirs qui aboutissent en dehors de la grille ou à des cases qui ont déjà subi des tirs). La fonction `new_who` permet d'alterner les tours de jeu des joueurs.

2 Initialisation du jeu

2.1 Paramétrage

Les types suivants sont définis dans le fichier `CPbattleship.ml` :

```
type t_matrix_param = {dx : int ; dy : int} ;;
type t_ships_param = {len : int ; sh_type : int array} ;;
type t_param = {mat : t_matrix_param ; ships : t_ships_param ;
                shoots_nb : int} ;;
```

Le type `t_matrix_param` permet de représenter les dimensions des grilles. Le type `t_ships_param` permet de représenter le nombre de bateaux des joueurs (champ `len`) et les types des bateaux, c'est-à-dire son nombre de parties pour chaque bateau : par exemple, le descriptif `{len = 5; sh_type = [|5 ; 4 ; 3 ; 3 ; 2|]}` indique que chaque joueur a cinq bateaux, occupant respectivement 5, 4, 3, 3 et 2 cases. Le type `t_param` permet de représenter les paramètres d'un jeu : en plus des informations précédentes, il contient le nombre de tirs que chaque joueur émet à chaque tour de jeu.

Question 1 Écrivez les fonctions `matrix_dx(prm : t_param) : int`, `matrix_dy(prm : t_param) : int`, `ships_number(prm : t_param) : int`, `ships_type(prm : t_param) : int`

`array` et `shoots_number(prm : t_param) : int` permettant de récupérer les valeurs contenues dans un paramètre.

Question 2 Écrivez la fonction `init_param() : t_param` dont le résultat est une valeur de paramètre permettant de jouer avec des grilles de taille `10 x 10`, 5 bateaux de tailles respectives 5, 4, 3, 3 et 2, et d'émettre 5 tirs à chaque tour de jeu (pour chaque joueur).

2.2 Tests de cases et voisinages

Question 3 Écrivez la fonction `is_in_mat(x, y, prm : int * int * t_param) : bool` indiquant si les coordonnées `x` et `y` correspondent à une case d'une grille.

Question 4 Écrivez les fonctions `is_empty(x, y, m : int * int * t_matrix) : bool` (resp. `is_ship(x, y, m : int * int * t_matrix) : bool`) permettant de savoir si une case de coordonnées `x` et `y` (supposées valides) est vide (resp. contient une partie de bateau).

Question 5 Écrivez la fonction `valid_empty_position(x, y, m, prm : int * int * t_matrix * t_param) : bool` qui teste si les coordonnées `x` et `y` correspondent à une case vide de la matrice `m`. En parallèle, écrivez de fonctions de test (test fonctionnel). Testez la fonction.

Question 6 Écrivez la fonction `is_empty_neighbour(x, y, m, prm : int * int * t_matrix * t_param) : bool` permettant de savoir si les `3 x 3` cases (si elles existent) centrées en `(x, y)` sont vides ou non (les coordonnées `x` et `y` sont supposées correspondre à une case d'une grille). En fait, cette fonction est utilisée dans le choix aléatoire de la position de bateaux, et l'objectif de la fonction est de tester s'il n'y a pas de bateaux dans le voisinage immédiat d'une case de coordonnées `(x, y)`. En parallèle, écrivez de fonctions de test (test fonctionnel). Testez la fonction. Faites attention aux cas où la case de coordonnées `(x, y)` est au bord de la grille (c'est-à-dire aux cas où toutes les cases du voisinage de la case `(x, y)` n'appartiennent pas à la grille).

Question 7 Écrivez la fonction `valid_empty_neighbour(x, y, m, prm : int * int * t_matrix * t_param) : bool` qui appelle la fonction précédente après avoir testé la validité des coordonnées `x` et `y`.

2.3 Calcul et insertion d'un bateau aléatoire

Le principe de l'insertion d'un bateau dans une grille est le suivant. Une case et une direction sont choisies aléatoirement, puis une liste de coordonnées est calculée, qui commence à la case choisie et se déploie dans la direction choisie. Si la liste de coordonnées est valide, c'est-à-dire qu'aucune case n'a une partie de bateau dans son voisinage, la liste décrit un nouveau bateau, qui est inséré dans la matrice d'un joueur.

NB. Il existe 8 directions possibles : vers la droite (codée `0`), en bas à droite (codée `1`), vers le bas (codée `2`), en bas à gauche (codée `3`), vers la gauche (codée `4`), en haut à gauche (codée `5`), vers le haut (codée `6`), en haut à droite (codée `7`).

Question 8 Écrivez une fonction `next_position(pt, dir : t_position * int) : t_position` qui calcule les coordonnées de la case suivant la case de coordonnées `pt` dans la direction `dir`.

Question 9 Écrivez une fonction *récursive* `insert_ship_matrix(l, m : t_pos_list * t_matrix) : unit` qui insère le nouveau bateau décrit par la liste `l` dans la grille `m`. Écrivez des fonctions de test (test structurel) pour cette fonction, et testez-la.

Question 10 Écrivez une fonction *itérative* `insert_rand_ship(nb_pos, m, l, prm : int * t_matrix * t_remaining_pos * t_param) : unit`, qui calcule et insère un bateau de taille `nb_pos` dans la grille `m`. Plus précisément, cette fonction :

1. calcule un bateau aléatoire de taille `nb_pos`, et vérifie sa validité, c'est-à-dire que chaque case du bateau est bien une case de la matrice, dont le voisinage ne contient aucune case d'un autre bateau. Une manière de faire consiste à boucler tant qu'un tel bateau n'a pas été calculé et, à chaque tour de boucle, de choisir aléatoirement une case et une direction, et à essayer de calculer un bateau valide commençant en cette case et se déployant dans la direction choisie ;

2. insère le nouveau bateau valide dans la grille `m` ;
3. « ajoute » les coordonnées des cases du nouveau bateau à la liste contenue dans le paramètre `l`. Celui-ci contient au début la liste des coordonnées des cases de tous les bateaux contenus dans la grille `m`, il doit en être de même à la fin de l'exécution de la fonction.

En parallèle, écrivez des fonctions de test (test fonctionnel), puis testez votre fonction.

2.4 Initialisation de la grille d'un joueur

Question 11 Écrivez une fonction *itérative* `init_matrix_ships_computer(prm : t_param) : t_matrix * t_remaining_pos` qui calcule une grille contenant des bateaux aléatoires de tous les types décrits dans le paramètre `prm`, ainsi qu'un élément mutable contenant la liste des coordonnées de toutes les cases de tous les bateaux de la grille. En parallèle, écrivez des fonctions de test (test fonctionnel), puis testez votre fonction.

Question 12 Écrivez une fonction `init_matrix_ships_human(myname, prm : string * t_param) : t_matrix * t_remaining_pos` qui calcule une grille vide ne contenant aucun bateau, et un élément mutable contenant une liste vide de coordonnées.

Question 13 Écrivez une fonction `init_matrix_ships(myname, hum, prm : string * bool * t_param) : t_matrix * t_remaining_pos` qui appelle l'une des deux fonctions précédentes suivant la valeur du paramètre `hum` indiquant si le joueur dont on initialise la grille est humain ou simulé.

2.5 Initialisation des joueurs et du jeu

Question 14 Écrivez une fonction *itérative* `count_positions_ship(prm : t_param) : int` qui calcule le nombre de cases composant l'ensemble des bateaux prévu par le paramètre `prm`.

Question 15 Écrivez une fonction `init_player(myname, hum, prm : string * bool * t_param) : t_player` qui initialise un joueur de nom `myname`, humain ou simulé suivant la valeur du paramètre `hum`.

Question 16 Écrivez une fonction `init_play(nm1, hum1, nm2, hum2, prm : string * bool * string * bool * t_param) : t_play` qui initialise les deux joueurs de noms respectifs `nm1` et `nm2`, humains ou simulés suivant les valeurs de `hum1` et `hum2`.

Au besoin, vous pouvez commencer à utiliser les fonctions d'affichage `draw_play` (pour un affichage graphique, après avoir ouvert une fenêtre graphique) ou `print_play` (pour un affichage textuel).

3 Tours de jeu

3.1 Choix de positions de tirs (aléatoires)

Question 17 Écrivez une fonction *récursive* `position_in_list(p, l : t_position * t_pos_list) : bool` qui teste si une position `p` appartient à une liste de positions `l`. Ecrivez des fonctions de test (test structurel). Testez votre fonction.

Question 18 Écrivez une fonction `rand_position(prm : t_param) : t_position` qui calcule une position aléatoire correspondant à une case d'une grille.

Question 19 Écrivez une fonction *récursive* `choose_one_shoot(l, m, prm : t_pos_list * t_matrix * t_param) : t_position` qui calcule une position aléatoire correspondant à une case vide d'une grille, et qui n'apparaît pas dans la liste `l`.

Question 20 Écrivez une fonction *récursive* `choose_shoots(nb_shoots, m, prm : int * t_matrix * t_param) : t_pos_list` qui calcule une liste de destinations de tirs aléatoires distinctes (les destinations de tirs doivent correspondre à des cases vides de la matrice `m`). En parallèle, écrivez des fonctions de test (test fonctionnel). Testez votre fonction.

3.2 Envoi et effet des tirs

Question 21 Écrivez une fonction *récursive* `rem_position_from_list(pos, l : t_position * t_pos_list) : t_pos_list` qui calcule une liste égale à la liste `l` privée de la position `pos`. En parallèle, écrivez des fonctions de test (test fonctionnel). Testez votre fonction.

Question 22 Écrivez une fonction `one_shoot(pos, player_1, player_2, prm : t_position * t_player * t_player * t_param) : bool` qui simule le tir du joueur `player_1` sur la case de coordonnées `pos` de la grille du joueur `player_2` (NB. on suppose que les coordonnées n'ont pas été vérifiées au préalable ; dit autrement, elles peuvent être quelconques, voir invalides par rapport à la taille de la grille). La fonction met à jour les informations concernées dans les descriptifs des deux joueurs (attention à ne rien oublier), en tenant compte bien sûr de l'arrivée du tir. Le résultat de la fonction est un booléen qui indique si le tir est correct ou non (le tir est incorrect s'il aboutit en dehors de la grille, ou s'il a eu pour destination une case sur laquelle le joueur avait déjà tiré).

Question 23 Écrivez une fonction *récursive* `all_shoots(shoots, player_1, player_2 : t_pos_list * t_player * t_player) : int`, qui simule l'ensemble des tirs effectués par le joueur `player_1` sur les cases de la grille du joueur `player_2`, dont les coordonnées sont décrites dans la liste `shoots` (comme précédemment, on suppose que les coordonnées n'ont pas été vérifiées au préalable). En parallèle, écrivez des fonctions de test (test fonctionnel). Testez votre fonction.

3.3 Un tour de jeu

Les fonctions ci-dessous sont écrites en supposant que c'est au tour du joueur `player_1` de jouer.

Question 24 Écrivez une fonction `play_one_player_computer(player_1, player_2, prm : t_player * t_player * t_param) : int`, qui effectue un tour de jeu, dans le cas où le joueur `player_1` est un joueur simulé. Le résultat est le nombre de tirs erronés (qui devrait être systématiquement nul si les fonctions précédentes sont correctes).

Question 25 Écrivez une fonction `let play_one_player_human(player_1, player_2, prm : t_player * t_player * t_param) : int` qui, pour l'instant, ne fait rien sauf retourner la valeur `0`.

Question 26 Écrivez une fonction `play_one_player(player_1, player_2, prm : t_player * t_player * t_param) : int` qui simule un tour de jeu où c'est au tour du joueur `player_1` de jouer, selon que ce joueur soit simulé ou humain.

Question 27 Écrivez une fonction `new_who(w : int) : int`, qui prend en paramètre le numéro du joueur qui vient de jouer (c'est-à-dire `1` ou `2`) et qui retourne le numéro du joueur dont c'est maintenant le tour de jouer.

La version de base du jeu est maintenant terminée, vérifiez que tout se passe bien à l'exécution.

4 Extensions

4.1 Extensions obligatoires

4.1.1 Tenir le compte des bateaux non détruits

Le champ `remain` d'un joueur contient la liste des coordonnées des cases de la grille du joueur correspondant à des parties intactes de ses bateaux. Mais en fait, cette simple liste ne permet de savoir directement quels sont les bateaux non détruits du joueur.

Question 28 En vous inspirant des types `t_remaining_pos` et `t_ships_param`, imaginez un type pour le champ `remain` qui permettrait de tenir à jour les descriptifs des bateaux d'un joueur.

Question 29 Modifiez les fonctions concernées afin de gérer ce nouveau descriptif.

De la même manière, le champ `opp_nb` permet de connaître le nombre de parties intactes des bateaux de son adversaire, mais ne permet pas directement de connaître les bateaux détruits de son adversaire.

Question 30 En vous inspirant du type `t_ships_param`, remplacez le champ `opp_nb` par un champ `opp_remain` permettant de mieux décrire ce qu'il reste à l'adversaire.

Question 31 Modifiez les fonctions concernées afin de gérer ce nouveau descriptif.

4.1.2 Faire jouer un joueur humain

Question 32 Comment imaginez-vous les fonctionnalités nécessaires pour faire jouer un joueur humain ?

Question 33 Affinez votre réflexion afin de décrire précisément ces fonctionnalités, et discutez-en avec votre enseignant.

Question 34 Une fois vos choix validés, concevez, développez et testez les fonctions nécessaires pour ces fonctionnalités.

4.2 Autres extensions

Vous avez maintenant toute latitude pour imaginer et réaliser de nouvelles fonctionnalités, par exemple :

- concevoir une meilleure statégie de jeu pour un joueur simulé ;
- améliorer l'interaction en développant certaines possibilités, par exemple en permettant de donner le résultat de chaque coup tiré au fur et à mesure, ou le résultat global d'un salve de plusieurs tirs, etc. Par exemple, grâce à la première extension obligatoire, chaque joueur doit pouvoir indiquer à l'autre le type d'un bateau détruit lors d'un tir sur la dernière partie non détruite de celui-ci. Vous pouvez bien sûr vous servir de la zone d'affichage de messages sur la fenêtre graphique ;
- distinguer plusieurs armes aux effets différents ;
- ajouter la possibilité de réparer des bateaux ;
- améliorer l'affichage graphique ;
- etc.

Question 35 . Précisez les fonctionnalités que vous souhaitez développer, et discutez-en avec votre enseignant. Puis concevez, développez et testez les fonctions nécessaires.