Projet Bataille Navale

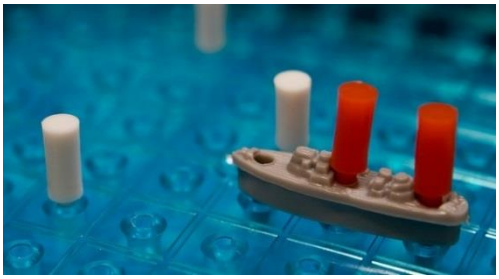
Les prochaines séances seront consacrées à un mini-projet de programmation autour du jeu de bataille navale.

Ce mini-projet vous permettra de revoir plusieurs concepts abordés cette année et sera pour vous l'occasion de découvrir différents aspects de la programmation : conception, implémentation, ajout de nouvelles fonctionnalités, etc.

En fonction de vos idées et de l'avancement du projet, plusieurs modules additionnels pourront être greffés au concept initial : création d'une intelligence artificielle, communications réseau, amélioration de l'interface graphique, ou encore parties à plus de 2 joueurs.

I. Règles du jeu

La bataille navale est un jeu qui opposent deux joueurs qui contrôlent chacun une flotte de navires et dont l'objectif est d'envoyer par le fond les bâtiments adverses avant de subir le même sort.



Au début de la partie, chaque joueur dispose sur une grille tenue secrète les positions de ses navires.

Dans la version la plus courante du jeu, la grille est un carré de 10 cases de côté, et chaque flotte est constituée de 5 bâtiments de différentes longueurs :

- Un porte-avions (5 cases de long)
- Un croiseur (4 cases)
- Un contre-torpilleur (3 cases)
- Un sous-marin (3 cases)
- Un torpilleur (2 cases)

Une fois les flottes positionnées, les hostilités commencent. A tour de rôle, chacun "tire" sur une case adverse. Plusieurs cas de figure sont possibles :

- *Raté* : il n'y a aucun bateau sur la case cible
- *Touché* : il y a un bateau sur la case cible, dont toutes les cases n'ont pas été touchées
- *Coulé* : il y a un bateau sur la case cible, dont la dernière case vient d'être touchée

Le gagnant est le premier joueur qui parvient à découvrir et à couler l'intégralité de la flotte adverse.

II. Conception

La première étape d'un projet de programmation consiste à déterminer quelles sont les données que l'on souhaite représenter, et quels sont les outils les plus adaptés pour cela.

II.a. Représentation des données

Question 1. Quelle(s) structure(s) de données proposeriez-vous pour représenter le jeu de bataille navale ? Pourquoi ?

Voici quelques pistes qui pourront aiguiller votre choix :

- Quelles sont les données à représenter pour chaque joueur ?
- Quelles sont les données susceptibles d'être modifiées ?
- Quels sont les "compteurs" qu'il faut mettre en place ?

II.b. Les types personnalisés en OCaml

Comme beaucoup de langages de programmation, OCaml offre la possibilité de créer des types personnalisés : nous allons donc pouvoir créer des objets qui vont répondre exactement aux besoins identifiés à la question précédente.

Déclarer un nouveau type

On peut notamment créer un type contenant plusieurs sous-objets. On utilise pour cela la syntaxe suivante[1]:

```
type nom_du_nouveau_type =
    {
      sous_objet_1 : type_1 ;
      sous_objet_2 : type_2 ;
      ...
      sous_objet_N : type_N ;
   };;
```

Par exemple, le type `personne` défini ci-dessous contient deux sous-objets : une chaine de caractère nommée `prenom` et un entier nommé `age` :

```
type personne =

    {
      prenom : string ;
      age : int
     };;
```

[1] Remarque : Un tel type est appelé un "type enregistrement". Il existe également les "types somme" et les "types abréviation", mais nous ne nous en servirons pas ici.

Utiliser un nouveau type

Une fois ce type déclaré, on peut créer des objets de ce nouveau type grâce la syntaxe suivante :

```
let objet_du_nouveau_type =
    {
      sous_objet_1 = val_1 ;
      sous_objet_2 = val_2 ;
      ...
      sous_objet_N = val_N ;
   };;
```

Par exemple,

```
let quelqu_un =

    {
      prenom = "Toto" ;
      age = 7
     };;
```

Remarque : pour déclarer un nouveau type, on utilise le mot clef `type` et le symbole deux-points (`:`), et pour utiliser ce nouveau type, on utilise le mot clef `let` et le symbole égal (=).

Accès à l'un des objets contenus dans un type

Pour accéder à l'un des sous-objets contenus dans un tel objet, on utilise la syntaxe suivante :

```
nom_de_l_objet.nom_du_sous_objet
```

Par exemple, `quelqu_un.prenom` renvoie la chaine `"Toto"`.

Permettre la modification de certains champs

Par défaut, les sous-objets composants un objet ne sont pas modifiables : ils sont définis à la création de l'objet et ne peuvent pas être changés par la suite.

Pour permettre la modification d'un de ces sous-objets, on utilise le mot clef `mutable`:

```
type nom_du_nouveau_type =
    {
      objet1 : type1 ;
      mutable objet2 : type2 ;
      ...
      objetN : typeN ;
   };;
```

Par exemple, si l'on souhaite pouvoir modifier l'`age` d'une `personne`, on obtient

```
type personne =
     {
       prenom : string ;
       mutable age : int
      };;
```

Question 2. Créer un type `eleve` constitué des sous-objets suivants :

- Un prénom (chaine de caractère)
- Un nom (chaine de caractère)
- Un age (entier modifiable)
- Une liste de notes (liste modifiable de nombre réels)

Question 3. Imaginez une fonction `calcul_moyenne` :

- Prenant en argument un objet de type `eleve`
- Renvoyant un nombre réel
- Telle que `calcul_moyenne e` renvoie la moyenne des notes de l'élève `e`

Question 4. Créer un type `classe` constitué des sous-objets suivants :

- Un nom (chaine de caractère)
- Une liste d'élèves (liste modifiable d'élèves)

Question 5. Imaginez une fonction `ajout_eleve` :

- Prenant en argument un objet de type `eleve` et un objet de type `classe`
- Ne renvoyant rien
- Telle que `ajout_eleve e c` ajoute l'élève `e` à la classe `c`

Question 6. Imaginez une fonction `calcul_moyenne_classe` :

- Prenant en argument un objet de type `classe`
- Renvoyant un nombre réel
- Telle que `calcul_moyenne_classe c` renvoie la moyenne de la classe `c`

II.c. Application à la bataille navale

Question 7. Créer le(s) type(s) adapté(s) à la représentation d'une bataille navale

Remarque : Le(s) type(s) créés devront prendre compte tous les éléments identifiés au II.a.

III. Affichage basique

Dans le cadre de ce projet, vous aurez peut-être besoin d'afficher le contenu d'un tableau d'entiers.

Pour cela, vous pouvez récupérer le fichier source "Affichage basique" sur le support en ligne. Enregistrer ce fichier au moment endroit que votre document de travail, et ajouter à ce dernier la ligne de code suivante : `#use "affichage_basique.ml";;`

Vous pourrez alors taper `tracer_grille t` pour afficher le contenu du tableau `t`.

IV. Positionnement des flottes

La mise en place des flottes peut être décomposée en plusieurs petites fonctions, dont l'implémentation vous permettra de vous approprier les types nouvellement créés.

Question 8. Imaginez une fonction permettant d' "initialiser" un joueur, c'est-à-dire de créer la structure de données correspondant à un joueur n'ayant pas commencé à placer ses bateaux.

Question 9. Implémentez une fonction renvoyant la liste des cases occupées par un bateau.

Question 10. Ecrivez une fonction vérifiant si un joueur peut ajouter un bateau à un emplacement donné.

Indice : Identifiez les facteurs qui peuvent limiter la pose d'un nouveau bateau sur la grille

Question 11. Implémentez une fonction qui ajoute un bateau sur la grille d'un joueur.

Question 12. Pour finir, implémenter une fonction permettant à un joueur de positionner un à un tous les bateaux de sa flotte (en vérifiant à chaque fois la validité du placement du bateau).

V. Déroulement de la partie

Une fois les deux flottes positionnées, la partie peut commencer. Un tour de jeu se fait en plusieurs étapes :

- Le joueur actif annonce une case de la flotte adverse
- Le joueur adverse lui indique le résultat de son tir : *raté*, *touché*, ou *coulé*.
- Si le joueur adverse vient de perdre son dernier bateau, la partie est terminée. Sinon un nouveau tour commence.

Question 13. Ecrivez une fonction qui renvoie le résultat d'un tir sur une flotte

Indice : Comment savoir si un bateau est *touché* ou *coulé* ?

Question 14. Ecrivez une fonction indiquant si un joueur a encore au moins un bateau à flot.

Question 15. Ecrivez enfin une fonction :

- prenant en argument
  - un "joueur" dont c'est le tour
  - un "joueur adverse"
  - les coordonnées d'une attaque
- effectuant les actions suivantes :
  - vérifier que l'attaque est valide (c'est-à-dire que la case n'a pas déjà été ciblée)
  - donner le résultat de l'attaque (raté, touché ou coulé)
  - mettre à jour les structures de données.

Indice : Après une attaque, il faut mettre à jour à la fois la "flotte adverse" du joueur en cours, et la flotte du joueur adverse.

Question 16. Comment pourrait-on utiliser les fonctions précédentes pour gérer le déroulement d'une partie de bataille navale ?

Indice : La bataille navale se joue au tour par tour, jusqu'à la défaite d'un des joueurs.