

Énoncé TP1
Battleship : Navigation

Directives :

- Vous devez remettre votre travail pour le dimanche 15 juin à 23h59. Tous les retards sont acceptés sans pénalité jusqu'au lundi 16 juin à 7h59. La note de 0% vous sera automatiquement attribuée si vous n'avez pas remis votre travail avant 8h.
- Il s'agit d'un travail en équipe de maximum 3 étudiants. Vous pouvez réaliser le travail seul, mais il est recommandé de former une équipe de 2 ou 3 membres.
- Vous devez compléter le code $F\#$ qui vous est fourni en respectant les contraintes du paradigme de programmation fonctionnelle. Aucune librairie externe n'est autorisée. Vous êtes limités à la librairie standard $F\#$.
- Les livrables comprennent votre code, un rapport au format PDF et une courte vidéo de démonstration de votre travail.
- L'utilisation de l'IA générative est permise pour le code seulement, quoique fortement déconseillée.
- La pondération pour ce travail est de 10%. Les critères de correction sont présentés en annexe A.

Contexte :

Pour ce travail pratique, vous devez compléter les méthodes utilitaires d'un mini-jeu inspiré du jeu de table classique Battleship. Le programme est présenté sous la forme d'une application Web locale comprenant une interface utilisateur et une librairie utilitaire. Les utilisateurs peuvent créer leur propre tableau de jeu et lancer une partie contre l'ordinateur doté d'une certaine *intelligence*.

L'interface utilisateur est déjà complétée pour ce travail. L'application Web contient deux pages distinctes, *Fleet Deployment* et *Play*, permettant respectivement de configurer un tableau personnalisé et de jouer une partie. Votre travail est de compléter seulement l'implémentation des méthodes utilitaires $F\#$. Ces méthodes sont appelées directement par l'interface utilisateur.

La première partie de ce projet vise seulement les fonctionnalités générales de manipulation de la grille de jeu ainsi que la navigation globale des bateaux. La deuxième partie ajoutera les capacités spéciales des bateaux, l'*intelligence* et les actions nécessaires au bon déroulement du jeu.

Documentation :

- **Fleet Deployment.**

La page d'accueil, qui est la page permettant de jouer une partie, vous demande d'abord d'importer le tableau de configuration désiré avant de lancer la partie. L'utilisateur peut créer son propre tableau personnalisé via la page *Fleet Deployment*. Un tableau réfléchi et construit avec soin peut aussi être un excellent outil pour tester vos implémentations.

1. **Grille** : La grille de jeu est un tableau en deux dimensions de coordonnées (i, j) . Pour une grille de 5 rangées et 10 colonnes, les coordonnées sont arrangées dans l'ordre suivant :

(0, 0)	(0, 1)	(0, 2)	(0, 3)	(0, 4)	(0, 5)	(0, 6)	(0, 7)	(0, 8)	(0, 9)
(1, 0)	(1, 1)	(1, 2)	(1, 3)	(1, 4)	(1, 5)	(1, 6)	(1, 7)	(1, 8)	(1, 9)
(2, 0)	(2, 1)	(2, 2)	(2, 3)	(2, 4)	(2, 5)	(2, 6)	(2, 7)	(2, 8)	(2, 9)
(3, 0)	(3, 1)	(3, 2)	(3, 3)	(3, 4)	(3, 5)	(3, 6)	(3, 7)	(3, 8)	(3, 9)
(4, 0)	(4, 1)	(4, 2)	(4, 3)	(4, 4)	(4, 5)	(4, 6)	(4, 7)	(4, 8)	(4, 9)

Du côté de la librairie $F\#$, la grille est définie dans le module **Grid** par un type polymorphe inductif qui contient un certain nombre de rangées, où chaque rangée est une liste de cellules.

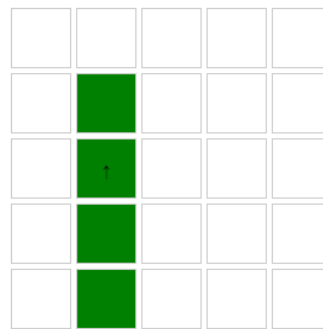
Dans cette première partie du projet, vous travaillerez principalement avec une grille de secteurs. Un secteur peut être vide ou bien actif. Un secteur actif contient des informations sur le bateau qui l'occupe.

2. **Classe** : Chaque bateau appartient à une certaine classe qui définit ses propriétés générales. Un bateau est unique, ce qui signifie qu'il ne peut pas exister deux bateaux de la même classe. Pour mieux visualiser les classes de bateaux, chaque classe est également associée à une couleur.
 - **Rouge** : Classe "Spy". Petit bateau avec une taille de seulement 2 cellules. Il s'agit du seul bateau allié contrôlé par le joueur.
 - **Lime** : Classe "PatrolBoat". Aussi un petit bateau d'une taille de 2 cellules. Ce bateau a le pouvoir de réparer d'autres bateaux et ainsi récupérer des points de vie.
 - **Aqua** : Classe "Destroyer". Bateau d'une taille moyenne de 3 cellules. La principale responsabilité de ce bateau est d'attaquer le bateau du joueur à l'aide de missiles.

- **Bleu** : Classe “Submarine”. Partage aussi une taille de 3 cellules. Ce bateau très puissant lance des torpilles sur la grille de jeu pour déconcentrer le joueur.
- **Vert** : Classe “Cruiser”. Grand bateau qui s’étend sur 4 cellules. La capacité spéciale de ce bateau permet d’intercepter des missiles destinés à d’autres bateaux et d’encaisser le choc à leur place.
- **Violet** : Classe “AircraftCarrier”. Très grand bateau qui occupe 5 cellules. Ce bateau sert de centre de commande pour diriger et contrôler les avions.

3. **Bateau** : Un bateau correspond à un type enregistrement englobant une classe, une liste de coordonnées, un centre de rotation et une direction (nord, sud, est, ouest).

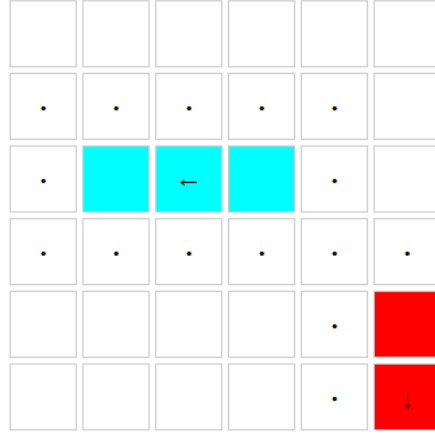
Les coordonnées d’un bateau doivent toujours être ordonnées de l’avant du bateau (index 0) vers l’arrière du bateau (index $n - 1$). L’avant du bateau est la première cellule pointée par la direction dans laquelle le bateau fait face. La direction d’un bateau est affichée sur la cellule désignée par le centre.



Dans l’exemple ci-haut, les coordonnées du bateau sont $[(1, 1), (2, 1), (3, 1), (4, 1)]$ et son centre est $(2, 1)$. La direction de ce bateau est le nord.

Seuls ces 4 secteurs de la grille sont actifs. La cellule $(1, 1)$ contient un secteur actif avec comme information $(Cruiser, 0)$, car il s’agit de la première coordonnée d’un bateau de classe “Cruiser”. De la même manière, la cellule $(3, 1)$ est un secteur actif qui contient la coordonnée numéro 2 du bateau “Cruiser”.

4. **Périmètre** : Une zone de sécurité autour de chaque bateau, appelée périmètre, restreint le placement des bateaux sur la grille. Le périmètre d’un bateau consiste en une bordure rectangulaire d’une épaisseur d’une seule cellule autour de ce dernier. Aucun bateau ne peut être placé dans cette zone.



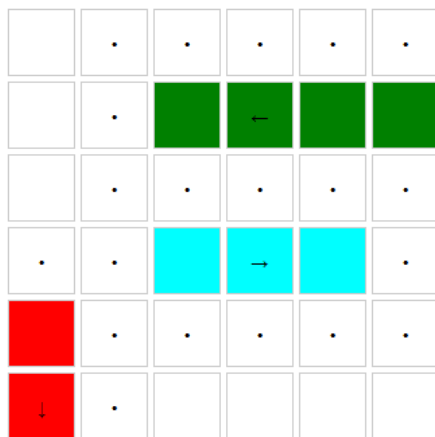
Dans l'exemple ci-haut, les périmètres sont illustrés par des points noirs sur la grille. Le périmètre du bateau aqua partage une cellule avec celui du bateau rouge. Les périmètres peuvent sans problème se superposer de cette manière, tant qu'aucun bateau n'occupe cet espace.

5. **Placement** : Les bateaux peuvent être placés sur la grille de manière individuelle. Puisque les coordonnées d'un bateau sont toujours ordonnées à partir de la tête, ses coordonnées sont déterminées en fonction de son centre et de l'orientation correspondant à la direction sélectionnée.

La position k du centre d'un bateau dans sa liste de coordonnées est obtenue en fonction de la formule suivante :

$$k = \left\lceil \frac{size}{2} \right\rceil - 1$$

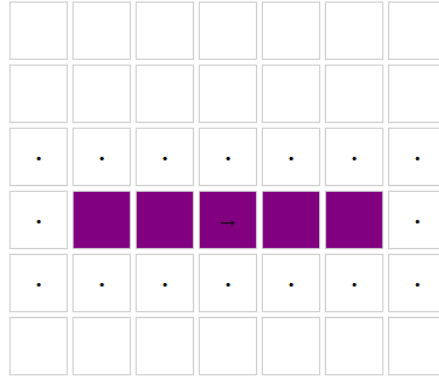
où $size$ dénote la taille du bateau présentement sous étude.



Dans l'exemple ci-haut, le bateau vert est placé dans la cellule (1, 3) dans la direction ouest, le bateau aqua se trouve dans la cellule (3, 3) orienté vers l'est, tandis que le bateau rouge est positionné dans la cellule (5, 0) et fait face au sud.

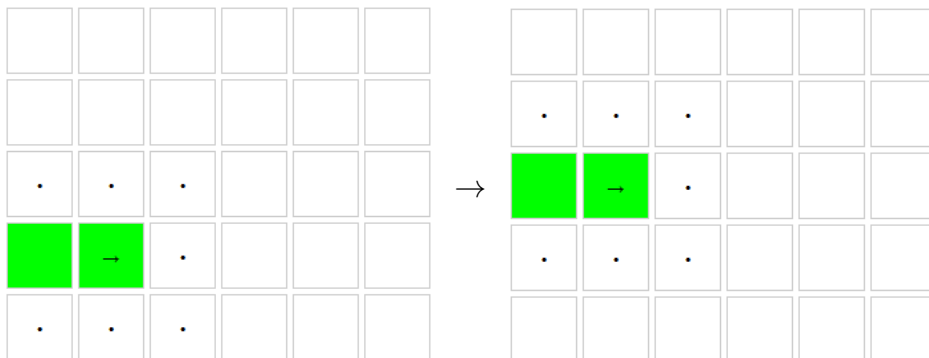
Pour qu'un placement soit valide, toutes les conditions suivantes doivent être respectées :

- Toutes les coordonnées du bateau doivent se retrouver à l'intérieur de la grille.
- Aucune coordonnée du bateau ne peut partager une cellule avec un autre bateau.
- Aucune coordonnée du bateau ne peut partager une cellule avec le périmètre d'un autre bateau.



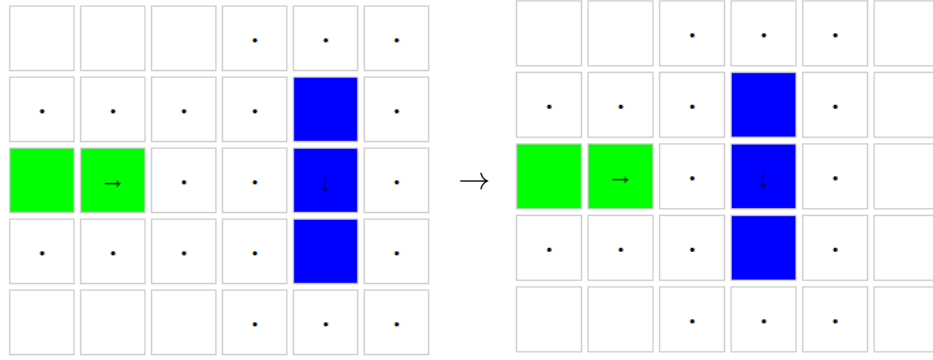
Dans l'exemple ci-haut, le bateau violet restreint fortement le placement des autres bateaux. Aucun bateau de taille 3 ne peut être placée en direction nord ou sud. Un bateau de taille 2 pourrait être orienté vers le nord seulement s'il est placé dans la première rangée (cellules $(0, j)$), ou bien faire face au sud uniquement en le plaçant dans la deuxième rangée (cellules $(1, j)$). Pour un bateau de taille 2 en direction est, les positions $(0, 0)$, $(1, 0)$ et $(5, 0)$ ne sont pas valides.

6. **Déplacement** : Pendant cette phase de configuration, il est possible de déplacer un bateau dans les 4 directions, peu importe l'orientation du bateau sélectionné. Dans l'exemple ci-bas, le bateau lime se déplace une fois vers le nord.



Pour qu'un déplacement soit valide, ses nouvelles coordonnées doivent respecter toutes les conditions de placement en considérant les autres bateaux :

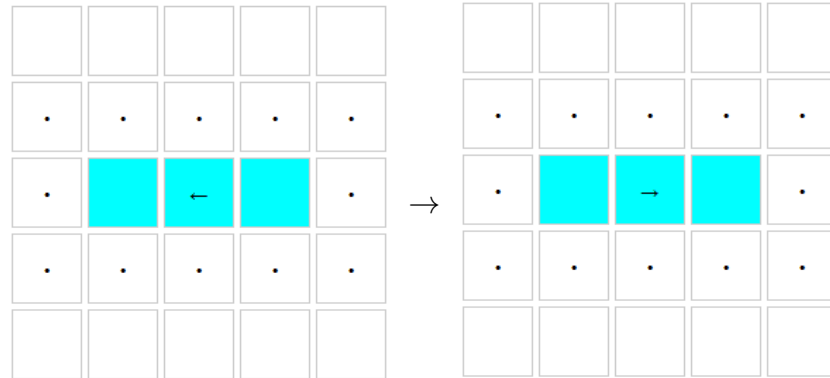
- Toutes les nouvelles coordonnées du bateau doivent se retrouver à l'intérieur de la grille.
- Aucune nouvelle coordonnée du bateau ne peut partager une cellule avec un autre bateau.
- Aucune nouvelle coordonnée du bateau ne peut partager une cellule avec le périmètre d'un autre bateau.



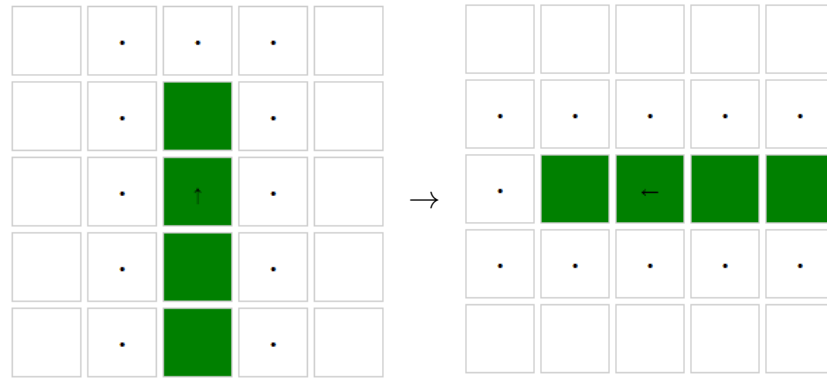
L'exemple ci-haut présente une autre situation de déplacement. Une fois que le bateau bleu se déplace vers l'ouest, les deux bateaux partagent 3 cellules pour former leur périmètre, ce qui signifie qu'ils ne peuvent plus se rapprocher davantage. Le bateau lime ne peut donc plus se déplacer ni vers l'est, ni vers l'ouest. Pour les mêmes raisons, le bateau bleu ne peut plus se déplacer vers l'ouest.

7. **Rotation** : Il est également possible de changer l'orientation d'un bateau après l'avoir placé. Le centre du bateau agit comme centre de rotation. Les contraintes habituelles sur l'ordre des coordonnées du bateau s'appliquent en tout temps.

Dans l'exemple ci-bas, le bateau aqua fait initialement face à l'ouest. Ses coordonnées sont $[(2, 1), (2, 2), (2, 3)]$ et son centre est $(2, 2)$. Après le changement de sa direction vers l'est, ses coordonnées sont maintenant dans l'ordre $[(2, 3), (2, 2), (2, 1)]$ et son centre demeure inchangé.

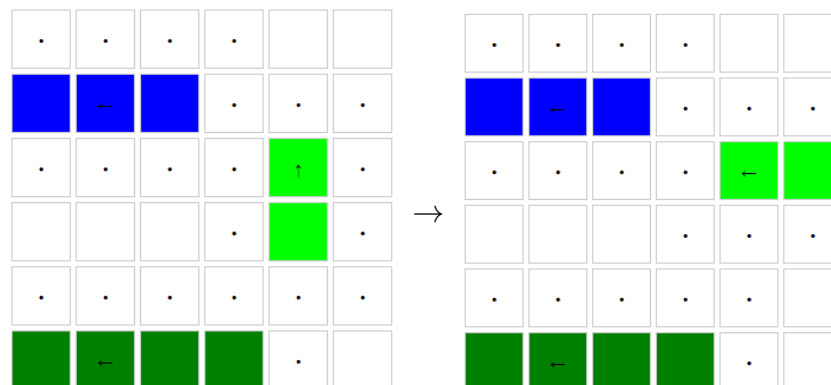


Le prochain exemple présente un changement de direction du nord vers l'ouest. Suite au changement, le centre est toujours situé dans la même cellule $(2, 2)$. Les coordonnées, qui étaient dans l'ordre $[(1, 2), (2, 2), (2, 3), (2, 4)]$, se retrouvent désormais dans l'ordre $[(2, 1), (2, 2), (2, 3), (2, 4)]$. Les coordonnées sont déterminées en fonction du centre du bateau, de la position du centre dans la liste de coordonnées et bien sûr de la direction.



De plus, il est important de respecter les conditions de placement d'un bateau suite à un changement de direction. Encore une fois, les nouvelles coordonnées doivent se conformer aux règles suivantes :

- Toutes les nouvelles coordonnées du bateau doivent se retrouver à l'intérieur de la grille.
- Aucune nouvelle coordonnée du bateau ne peut partager une cellule avec un autre bateau.
- Aucune nouvelle coordonnée du bateau ne peut partager une cellule avec le périmètre d'un autre bateau.



Dans la configuration ci-haut, l'orientation du bateau lime passe du nord à l'ouest. Ce dernier pourrait évidemment refaire face au nord et aussi au sud, mais pas à l'est, car son centre doit être en première position de la liste de coordonnées et le reste du bateau empiéterait sur le périmètre du bateau bleu. Pour sa part, le bateau bleu pourrait sans problème être orienté dans les 4 directions et le bateau forêt ne peut qu'alterner entre l'ouest et l'est.

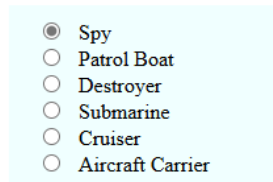
8. **Contrôles** : La configuration des bateaux débute par le choix des dimensions de la grille de jeu. Une fois que l'utilisateur valide les dimensions, les autres contrôles sont déverrouillés dans l'interface utilisateur. La grille de jeu est ensuite initialisée à l'aide de la fonction `Battlefield.initClearGrid`.

Rows :
Columns :

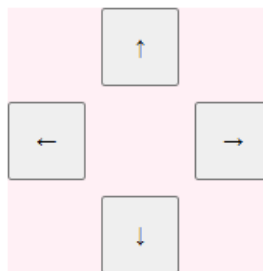
Lock

Le bateau sélectionné peut par après être placé sur la grille s'il respecte les conditions. À chaque mouvement de la souris sur la grille, la fonction `Navigation.canPlace` est appelée pour vérifier l'emplacement. Si la cellule respecte les contraintes, un click positionne définitivement le bateau en officialisant le tout avec les fonctions `Ship.createShip` et `Battlefield.addShip`.

Une fois positionné sur la grille, le périmètre d'un bateau est affiché en tout temps. La fonction `Ship.getPerimeter` procure à l'interface utilisateur les coordonnées touchées par le périmètre de chaque bateau existant.

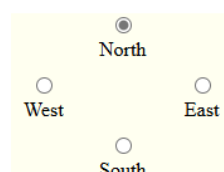


Si le bateau sélectionné est déjà placé sur la grille, le déplacement de la souris cherche plutôt à mettre en évidence le bateau survolé. La classe de ce bateau est déterminée avec la fonction `Battlefield.getSelectedName`. Un click de la souris sur un autre bateau permet de le sélectionner.



Les 4 boutons de navigation offrent la possibilité de déplacer le bateau sélectionné. Ce dernier doit être déjà placé sur la grille pour pouvoir le déplacer dans la direction souhaitée. La fonction `Navigation.canMove` indique si le déplacement est valide, auquel cas la fonction `Navigation.move` calcule les nouvelles coordonnées du bateau qui est finalement mis à jour dans la grille avec la fonction `Battlefield.replaceShip`.

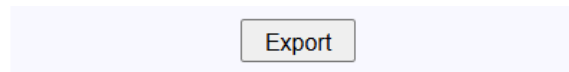
De manière similaire aux déplacements, le choix de l'orientation du bateau sélectionné requiert l'approbation par la fonction `Navigation.canRotate`. Si la rotation respecte les conditions, La fonction `Navigation.rotate` détermine les nouvelles coordonnées du bateau qui doit encore une fois être mis à jour dans la grille avec la fonction `Battlefield.replaceShip`.



Si le bateau sélectionné n'est pas encore présent sur la grille de jeu, il est possible de changer son orientation avec les mêmes contrôles. Dans ce cas, la direction initiale

est partagée aux fonctions `Navigation.canPlace` et `Ship.createShip` lorsque vient le temps de déposer un nouveau bateau sur la grille.

Un bouton spécial permet d'exporter la configuration créée jusqu'à maintenant à l'aide de la fonction `Battlefield.extractData`. Cette configuration contient les dimensions de la grille ainsi que les informations nécessaires sur les positions, directions et classes des bateaux existants. Le résultat est affiché au format JSON pour faciliter l'interprétation. L'utilisateur est invité à sauvegarder la configuration pour utilisation future.

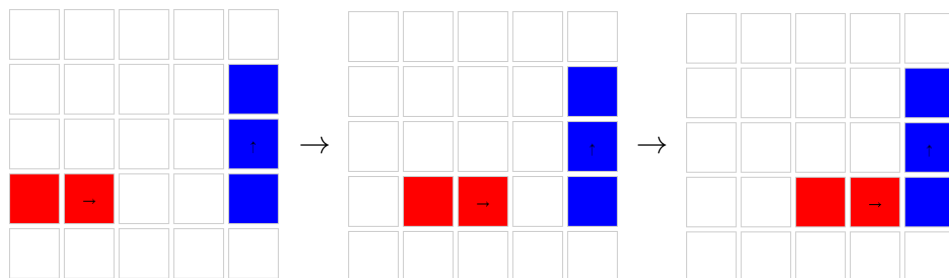


Les informations sont emmagasinées dans un type enregistrement `Data` qui contient les dimensions de la grille ainsi que la liste des bateaux présents sur la grille au moment de l'exportation des données.

- **Play.**

Il est maintenant temps de lancer une partie. Dans cette interprétation du jeu classique, l'utilisateur est le seul joueur. Un bateau espion, contrôlé par le joueur, se faufile pour gagner des avantages et attaquer les autres bateaux positionnés sur le plateau de jeu. Le joueur gagne la partie s'il réussit à couler tous les autres bateaux avant que son bateau espion ne perde tous ses points de vie.

1. **Déplacement vers l'avant** : En dehors de la phase de configuration, les bateaux ne peuvent se déplacer que dans la direction qui leur fait face. Le bateau espion se déplace vers l'avant à la demande du joueur.



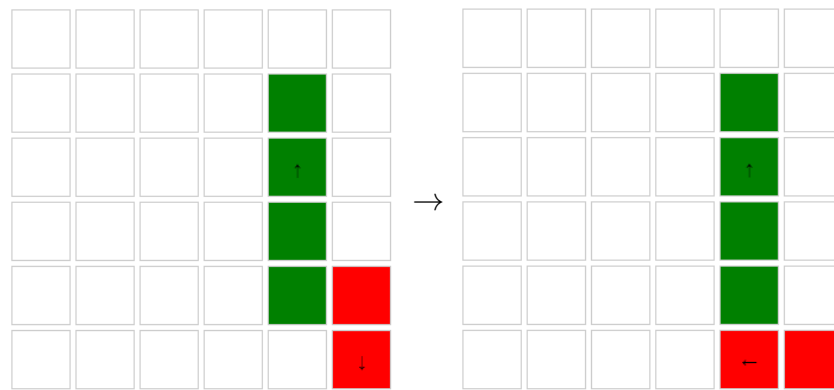
Dans l'exemple ci-haut, le bateau du joueur se déplace deux fois vers l'est, car il est orienté en direction de l'est. Notez que la notion de périmètre ne s'applique pas pendant le déroulement de la partie. Le bateau rouge ne peut toujours pas sortir de la grille ou passer par-dessus un autre bateau, mais il peut maintenant circuler librement dans son périmètre.

Plus formellement, un déplacement vers l'avant n'est valide que si les nouvelles coordonnées du bateau respectent toutes les conditions suivantes :

- Toutes les nouvelles coordonnées du bateau doivent se retrouver à l'intérieur de la grille.
- Aucune nouvelle coordonnée du bateau ne peut partager une cellule avec un autre bateau.

2. **Rotation vers l'avant** : Une rotation vers l'avant est en réalité une composition de deux mouvements : une rotation immédiatement suivie d'un déplacement vers l'avant dans la nouvelle direction. Une rotation peut s'effectuer dans le sens horaire ou bien antihoraire.

La nouvelle direction est déterminée en fonction de l'orientation courante du bateau et du sens de la rotation. Par exemple, la direction qui suit l'ouest dans le sens horaire est le nord alors que l'est vient après le sud dans le sens antihoraire.



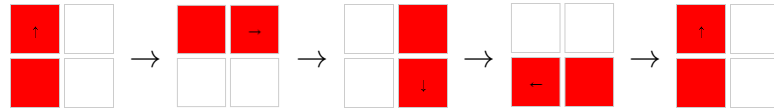
Dans l'exemple ci-haut, une rotation vers l'avant est appliqué sur le bateau rouge dans le sens horaire. La direction qui suit le sud dans le sens horaire est l'ouest, donc le bateau fait maintenant face à l'ouest. Un déplacement vers l'avant conclut le mouvement.

Cette séquence de déplacement n'aurait pas été possible en séparant les deux mouvements, car la rotation vers l'ouest aurait fait en sorte que la deuxième coordonnée du bateau ne respecte plus les contraintes de dimensions de la grille.

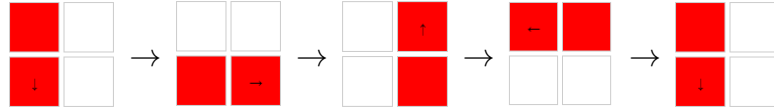
Tout comme les déplacements vers l'avant, une rotation vers l'avant ignore le périmètre des autres bateaux. Il faut toutefois s'assurer que toutes les nouvelles coordonnées respectent les conditions habituelles :

- Toutes les nouvelles coordonnées du bateau doivent se retrouver à l'intérieur de la grille.
- Aucune nouvelle coordonnée du bateau ne peut partager une cellule avec un autre bateau.

Appliquer plusieurs rotations vers l'avant consécutives dans le même sens permet de tourner en rond. Si le bateau rouge est limité à une grille de dimensions 2×2 pour naviguer, la seule séquence de mouvements possibles est la suivante :



La séquence ci-haut applique des rotations dans le sens horaire. L'équivalent dans le sens antihoraire correspond à ceci :

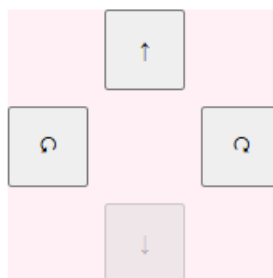


Encore une fois, ces mouvements n'auraient pas été possibles s'ils étaient effectués de manière séparée. La composition de la rotation et du déplacement vers l'avant permet de naviguer plus aisément à travers la grille de jeu.

3. **Contrôles** : La première partie de ce projet ne permet pas de jouer une partie complète. En fait, seuls les mouvements du bateau du joueur sont permis. L'utilisateur doit d'abord importer la configuration souhaitée pour créer la grille de jeu.

Une zone de texte est prête à recueillir les données au format JSON préalablement exportées. Le bouton accompagnateur fait appel à la fonction `Battlefield.loadData` pour reconstruire la grille de jeu à partir des données. Les autres contrôles sont ensuite déverrouillés.

Les seuls autres contrôles pour le moment sont des boutons qui permettent au joueur de naviguer son bateau espion à travers la grille. Les boutons reflètent le plus fidèlement possible le résultat de l'action, qui est soit un déplacement vers l'avant, soit une rotation vers l'avant. La fonction `Navigation.getNextDirection` est bien utile pour mettre à jour le texte des boutons de navigation.



Dépendamment de l'action que le joueur souhaite accomplir, une des deux fonctions `Navigation.canMoveForward` ou `Navigation.canRotateForward` est appelée pour valider le mouvement. Si toutes les conditions sont respectées, les fonctions

`Navigation.moveForward`, `Navigation.rotateForward` et `Battlefield.replaceShip` officialisent le mouvement.

Tâches :

0. Mise en place de l'environnement de développement.

Le projet est une application Web de type Blazor accompagnée d'une librairie *F#*. Vous êtes libres de choisir l'IDE qui vous convient. Dépendamment de l'IDE retenu, assurez-vous d'ajouter les installations nécessaires pour le développement Web ASP.NET et *F#*. Veuillez vous référer à la section *Matériel didactique* → *Logiciels* du site de cours pour plus d'informations sur le choix de l'IDE.

Vous avez à votre disposition une solution Battleship englobant deux projets :

- Battleship.Web : Projet de type “Blazor Web App” qui contient l'interface utilisateur et toute la logique de jeu.
- Battleship.Core : Projet de type “*F#* Class Library” qui contient toutes les fonctions utilitaires nécessaires au bon déroulement du jeu.

Le code de départ qui vous est fourni a été conçu avec le framework .NET 9.0. Vous n'avez rien à configurer pour être en mesure de lancer l'application en mode debug.

1. Fonctions.

La prochaine étape est de remplir les fonctions qui vous sont demandées. Pour ce faire, vous êtes limités à la partie “fonctionnelle” du langage *F#*, ce qui comprend essentiellement le filtrage par motifs, les fonctions et la récursion. Plus formellement, tout ce qui n'a été présenté dans le module “Programmation fonctionnelle et générique” est interdit.

La documentation plus haut présente les interactions possibles entre l'interface utilisateur et la librairie *F#*. Les fonctions à compléter sont réparties dans 3 modules :

~~• Ship:~~


– ~~createShip~~ (*center*: *Coord*) (*facing*: *Direction*) (*name*: *Name*) : *Ship*

Crée un bateau selon les paramètres reçus. Cette fonction doit calculer les coordonnées correspondant à la disposition du bateau. L'ordre des coordonnées est important et doit toujours refléter l'orientation.

– ~~getPerimeter~~ (*ship*: *Ship*) (*dims*: *Dims*) : *Coord list*

Détermine les coordonnées faisant partie du périmètre d'un bateau. L'ordre des coordonnées n'est pas important, mais toutes les coordonnées doivent respecter les dimensions de la grille.

- Navigation :

- ~~canPlace~~ (*center : Coord*) (*direction : Direction*) (*name : Name*) (*grid : Sector Grid*) : *bool*
Vérifie s'il est possible de placer un nouveau bateau sur la grille selon les conditions présentées précédemment.
- ~~canMove~~ (*ship: Ship*) (*direction: Direction*) (*grid: Sector Grid*) : *bool*
Vérifie s'il est possible de déplacer un bateau dans la direction spécifiée selon les conditions présentées précédemment.
- ~~move~~ (*ship: Ship*) (*direction: Direction*) : *Ship*
Produit un nouveau bateau déplacé d'une cellule dans la direction spécifiée.
- ~~canRotate~~ (*ship: Ship*) (*direction: Direction*) (*grid: Sector Grid*) : *bool*
Vérifie s'il est possible de changer l'orientation d'un bateau dans la direction spécifiée selon les conditions présentées précédemment.
-  ~~rotate~~ (*ship: Ship*) (*direction: Direction*) : *Ship*
Produit un nouveau bateau orienté dans la direction spécifié.
- *canMoveForward* (*ship: Ship*) (*grid: Sector Grid*) : *bool*
Vérifie s'il est possible de déplacer un bateau vers l'avant selon les conditions présentées précédemment.
- *moveForward* (*ship: Ship*) : *Ship*
Produit un nouveau bateau déplacé d'une cellule dans la direction qui lui fait face.
- *getNextDirection* (*current: Direction*) (*rotation: Rotation*) : *Direction*
Détermine la prochaine direction selon le sens de la rotation à appliquer.
- *canRotateForward* (*ship : Ship*) (*rotation : Rotation*) (*grid : Sector Grid*) : *bool*
Vérifie s'il est possible d'appliquer une rotation et avancer un bateau dans le même mouvement selon les conditions présentées précédemment.
- *rotateForward* (*ship: Ship*) (*rotation: Rotation*) : *Ship*
Produit un nouveau bateau orienté dans la prochaine direction selon la rotation spécifiée, puis ensuite déplacé d'une cellule dans cette nouvelle direction.

- Battlefield :

- ~~initClearGrid~~ (*dims: Dims*) : *Sector Grid*
Crée une nouvelle grille avec les dimensions spécifiées. Chaque cellule de la grille est un secteur vide.
- ~~addShip~~ (*ship: Ship*) (*grid: Sector Grid*) : *Sector Grid*
Produit une nouvelle grille où le bateau a été ajouté. Les secteurs occupés par ce

bateau sont maintenant actifs et contiennent le numéro de coordonnées associées.

- ~~replaceShip~~ (*ship: Ship*) (*grid: Sector Grid*) : *Sector Grid*
Produit une nouvelle grille où les secteurs occupés par le bateau sont mis à jour. Les secteurs qui ne sont plus occupés par le bateau sont à nouveau vides.
- ~~getSelectedName~~ (*coord: Coord*) (*grid: Sector Grid*) : *Name option*
Détermine le nom de la classe du bateau, possiblement inexistant, occupant la cellule visée dans la grille.
- *extractData* (*grid: Sector Grid*) : *Data*
Produit les données nécessaires pour exporter les dimensions de la grille et tous les bateaux qui s’y trouvent. Les bateaux sont reconstruits depuis la grille selon les secteurs occupés et leur numéro de coordonnées.
- *loadData* (*data: Data*) : *Sector Grid*
Crée une nouvelle grille selon les dimensions accessibles dans les données et importe tous les bateaux disponibles.

2. Abstraction.

Une fois que toutes vos implémentations sont complétées et que votre programme est fonctionnel, il est maintenant temps d’améliorer votre code à l’aide d’une couche d’abstraction supplémentaire. Vous aurez certainement constaté par vous-même que certains calculs sont souvent répétées d’une fonction à l’autre.

Vous devez isoler ces procédures répétitives dans des fonctions qui seront réutilisées à travers vos solutions. Présentez de manière informelle ces fonctions dans votre rapport en mentionnant leur rôle. Chaque nouvelle fonction peut être définie dans le module qui vous semble le mieux adapté. Attention de bien respecter l’ordre de dépendance entre les modules pour éviter des problèmes de dépendances circulaires.

On vous demande aussi d’identifier les comportement communs dans votre code et d’en extraire des fonctions d’ordre supérieur à définir dans le module **Grid**. Vos fonctions doivent évidemment manipuler des données de type polymorphe 'a **Grid**. Vous n’avez qu’à définir les fonctions qui vous sont utiles.

Présentez dans votre rapport les fonctions d’ordre supérieur définies dans le module **Grid** en spécifiant le type de chaque fonction. Discutez brièvement de votre cheminement pour identifier ces fonctions, les modifications nécessaires pour les intégrer dans vos solutions et les difficultés rencontrées.

Après ces étapes, votre code devrait être complètement épuré, sans aucune répétition de calcul ou de logique. Un bon découpage de fonctions facilitera le développement de la prochaine phase du travail.

3. Vidéo.

Une fois que votre travail est terminé, enregistrez une courte vidéo d'une durée maximale de 3 minutes sans aucune coupure ni transition. Aucune introduction ou conclusion n'est nécessaire.

Votre vidéo doit présenter chaque fonctionnalité du travail avec quelques petits exemples variés. Les fonctionnalités à inclure dans votre démonstration doivent apparaître généralement dans cet ordre :

(a) Création d'une configuration (*Fleet Deployment*) :

1. Générer une nouvelle grille.
2. Placer quelques bateaux dans des directions variées, en incluant des tentatives de placement invalide.
3. Déplacer quelques bateaux dans des directions variées, en incluant des tentatives de déplacement invalide.
4. Changer l'orientation de quelques bateaux dans des directions variées, en incluant des tentatives de rotation invalide.
5. Sélectionner quelques bateaux présents sur la grille.
6. Exporter la configuration.

(b) Lancement d'une partie (*Play*) :

7. Importer la configuration précédemment créée.
8. Déplacer et tourner le bateau rouge dans des directions variées, en incluant des tentatives de déplacement invalide.

Plusieurs points sont interchangeables. Assurez-vous seulement de bien segmenter votre vidéo pour d'abord promouvoir ce qui concerne la configuration du tableau, puis finalement la navigation dans une partie réelle.

Ajoutez simplement un lien vers votre vidéo dans l'en-tête de votre rapport.

4. Rapport.

Tous les éléments à insérer dans votre rapport sont décrits dans les sections précédentes. Pour résumer, on devrait retrouver dans votre rapport un lien vers votre vidéo ainsi que quelques explications et discussions à propos des fonctions élaborées pendant la phase d'abstraction.

Merci de retirer toutes formalités (page titre, table des matières, bibliographie et références, etc.) de votre rapport.

Remise :

Vous devez remettre dans la boîte de dépôt du portail un fichier ***.zip** qui contient l'entièreté de votre travail, y compris les fichiers que vous n'avez pas modifiés. Assurez-vous de remettre seulement le code sans les répertoires résiduels **bin**, **obj**, **.vs**, **.idea**, etc.

Vous devez également ajouter dans votre remise votre rapport au format PDF et le fichier **README.md** où vous indiquerez la version .NET, le système d'exploitation et l'IDE utilisés.

Bon travail !

Annexe A : Critères de correction.

Votre note finale pour ce travail est la somme des notes de chaque tâche, pour un minimum de 0/10 et un maximum de 10/10.

Tâche	Min	Max	Critère	Pénalité
Fonctions	-2	5	Pour chaque fonction invalide ou non complétée	-1
Abstraction	0	3	Effort insuffisant	-3
			Les fonctions définies ne sont pas réutilisées	-2
			Absence de fonctions d'ordre supérieur	-2
Vidéo	-2	1	Absente ou fortement incomplète	-3
			Qualité et présentation	-3
			Ne reflète pas le comportement du travail remis	-2
			Ne présente pas certaines fonctionnalités	-2
			Ne respecte pas la durée demandée	-1
Rapport	-2	1	Absent ou fortement incomplet	-3
			Qualité et présentation	-3
			Ne présente pas les fonctions ajoutées	-2
			Ne discute pas des difficultés rencontrées	-1
Général	-10	0	Le code remis ne compile pas	-10
			Erreur pendant l'exécution	-10
			Non respect du paradigme de programmation fonctionnelle	-10
			Modifications non autorisées	-10
			Qualité et lisibilité du code	-5
			Répertoires résiduels (<code>bin</code> , <code>obj</code> , <code>.vs</code> , <code>.idea</code>) inclus dans la remise	-2
			Fichier <code>README.md</code> absent ou non complété	-1