Langages de programmation
IFT-3000
E25

Énoncé TP2

Battleship : Gameplay

Directives :

- Vous devez remettre votre travail pour le dimanche 20 juillet à 23h59. Tous les retards sont acceptés sans pénalité jusqu'au lundi 21 juillet 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 des paradigmes de programmation fonctionnelle et impérative. 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 deuxième partie de ce projet vise le déroulement du jeu, ce qui inclut les fonctionnalités précises reliées aux interactions possibles avec les capacités spéciales des bateaux ainsi que l'*intelligence* de l'ordinateur. Le joueur peut entreprendre une action à chaque tour et l'ordinateur répond à cette action en exploitant son *intelligence*.

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.

  Il n'y a aucun changement dans cette page pour cette deuxième partie du travail. Veuillez vous référer à la documentation de la première partie pour les détails de cette page.

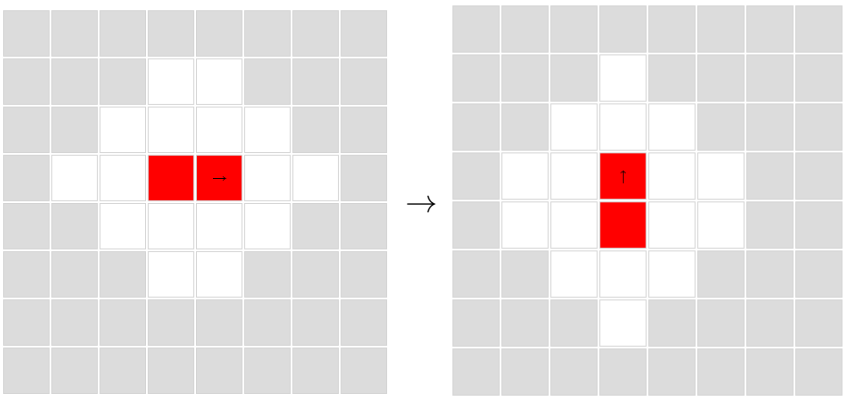
- **Play.**

  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 sur la grille pour éviter des obstacles 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.

  Tous les déplacements implémentés dans la première partie du travail sont encore valides. On ajoute à ces déplacements les capacités spéciales de chaque bateau. Les capacités spéciales sont contrôlées par l'*intelligence* qui tente d'appliquer une stratégie impitoyable pour vaincre le bateau espion.

  1. **Brouillard** : Seule une certaine zone de la grille est visible par le joueur. Le bateau espion est équipé d'un radar pour naviguer à travers le brouillard. Les cellules qui ne sont pas dégagées par le radar sont masquées derrière un épais brouillard.

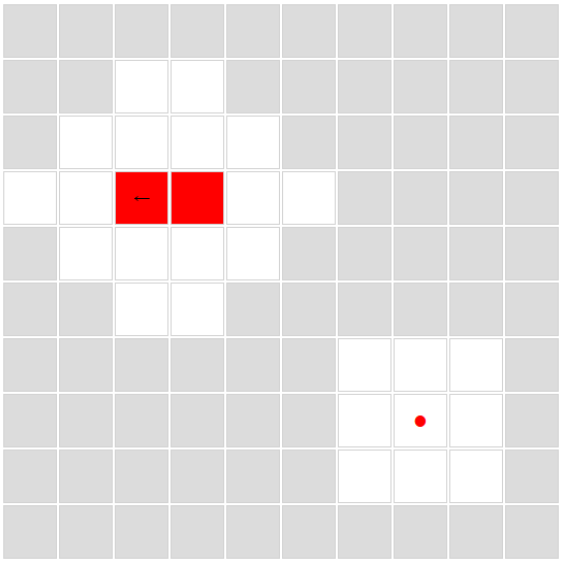
     Le radar prend la forme d'un losange centré entre les deux cellules occupées par le bateau espion. La position et l'orientation du bateau espion doivent évidemment être prises en compte pour déterminer les cellules incluses par le radar.

     Le brouillard est représenté dans le projet par une grille de booléens (`bool Grid`), où chaque cellule contient la valeur `false` si elle est recouverte de brouillard et `true`

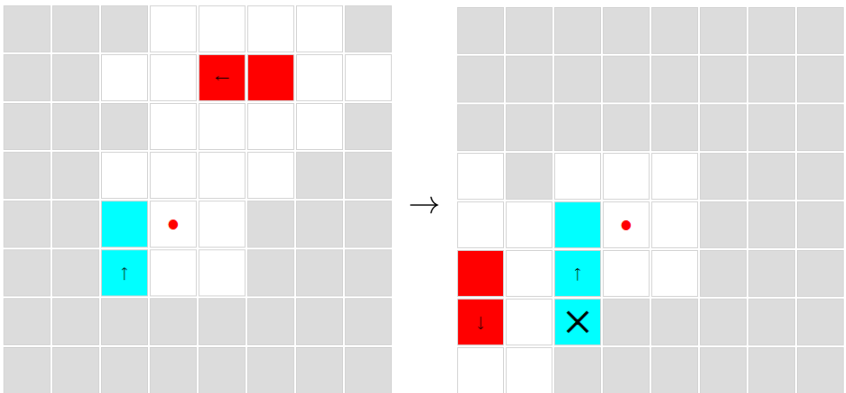
si elle est dégagée.

2. **Drone** : Le joueur peut également contrôler un drone. Ce dernier se déplace librement dans la grille, puisqu'il n'est pas soumis au trafic maritime. La seule utilité du drone est d'augmenter la vision du joueur avec une zone supplémentaire de 9 cellules centrées sur le drone lui-même. Le drone est visualisé avec un point rouge sur la grille.



Le radar du bateau et celui du drone peuvent sans problème se chevaucher. Le brouillard est donc toujours calculé en fonction du radar du bateau et du drone. Le drone ne peut pas être détruit par les bateaux adverses, ce qui le rend très puissant tout au long de la partie.

3. **Missile** : Le joueur peut lancer un missile sur les bateaux adverses afin de leur retirer des points de vie. Il n'y a aucune restriction sur le nombre total de missiles lancés durant une partie. Un missile peut seulement être lancé sur une cellule qui est dégagée par le radar du bateau espion. Une cellule qui est visible uniquement grâce au drone ne peut pas être visée par un missile.



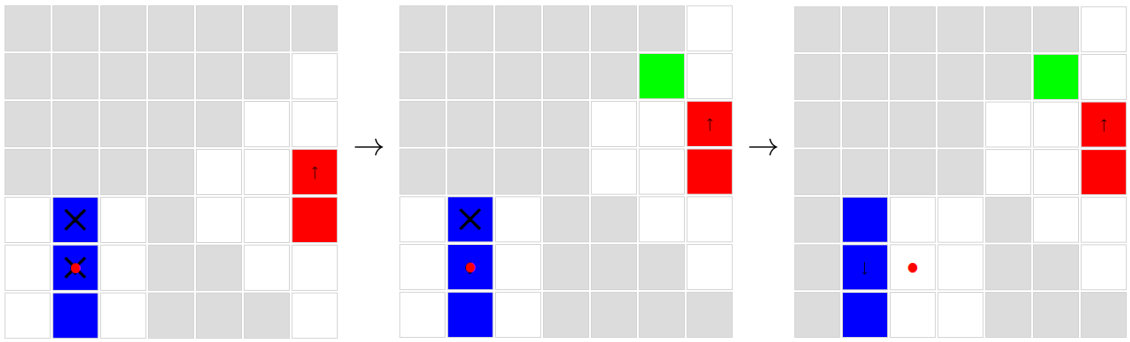
Dans la situation ci-haut, le joueur repère un bateau adverse à l'aide du drone. Le joueur doit ensuite se déplacer jusqu'à ce que le bateau aqua se trouve dans la zone

dégagée par le radar du bateau espion. Il est ensuite possible de lancer un missile sur une cellule occupée par le bateau adverse et ainsi lui retirer un point de vie.

4. **PatrolBoat** : Lorsque le bateau espion se situe dans le périmètre du bateau lime, la capacité spéciale de ce dernier est activée. Il est alors possible de réparer un bateau endommagé. La vérification se fait après chaque déplacement du bateau espion ou bien du drone.

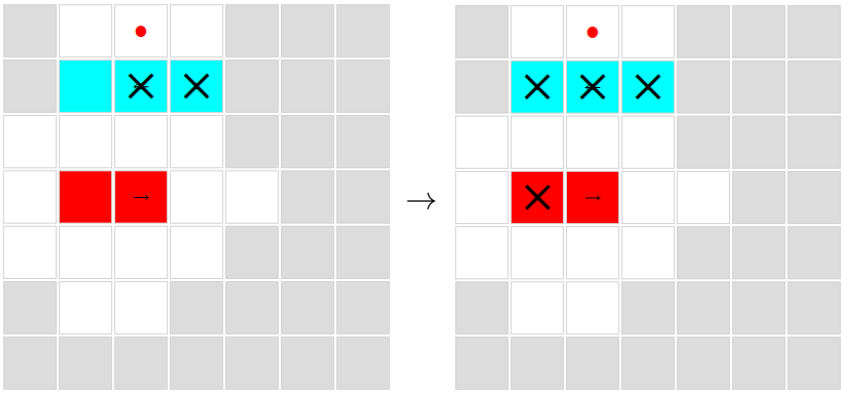
   La réparation permet de redonner un point de vie à un bateau au choix de l'*intelligence*. Le bateau sélectionné ne peut pas être un bateau préalablement coulé. Si aucun bateau ne peut être réparé, alors la réparation n'a simplement pas lieu. La cellule qui fait l'objet d'une réparation ne doit pas nécessairement être dégagée par le radar du bateau espion.

   Dans l'exemple ci-bas, dès que le bateau espion entre dans le périmètre du bateau lime, une réparation est effectuée. L'*intelligence* choisi ici de réparer le bateau bleu. Le joueur déplace ensuite le drone. Puisque le bateau espion se trouve toujours dans le périmètre du bateau lime, une deuxième réparation est effectuée.



   Au moment où le bateau lime lui-même est coulé, le bateau espion reçoit une réparation. Cette réparation précieuse peut spontanément changer l'issue d'une partie en raison des points de vie limités du bateau du joueur.

5. **Destroyer** : Après la dernière attaque qui permet de couler le bateau aqua, il lance un missile sur le bateau espion, lui retirant immédiatement un point de vie. Le choix de la cellule visée par ce missile de dernier souffle est réservée à l'*intelligence*.

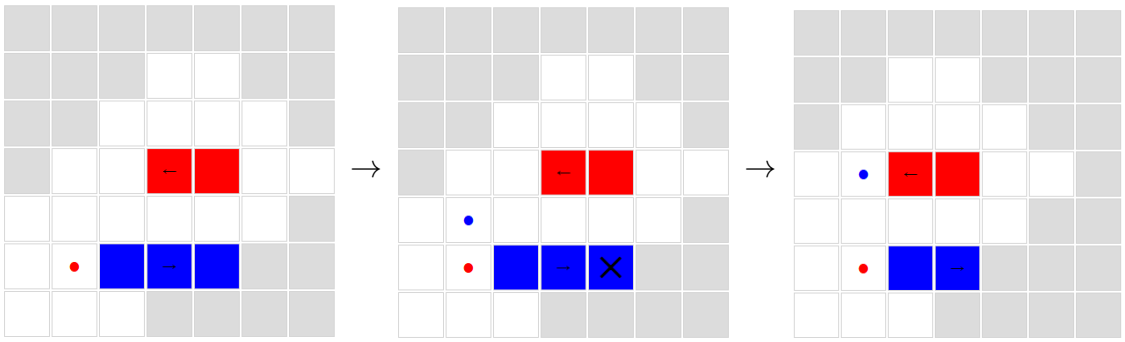


Dans le jeu ci-haut, le bateau aqua a déjà perdu deux points de vie. Le joueur décide de lancer un dernier missile sur ce bateau pour le couler, ce qui active sa capacité spéciale. L'*intelligence* choisi ici de lancer en retour un missile sur l'arrière du bateau espion.

6. **Submarine** : À chaque fois que le bateau bleu est atteint par un missile, il déploie une nouvelle torpille sur la grille. La torpille est une nouvelle entité qui occupe une cellule de la grille de jeu représentée par un point bleu. Elle ne peut donc pas cohabiter avec un bateau sur la même cellule.

   La position initiale de chaque nouvelle torpille doit se trouver dans le périmètre du bateau bleu. Après un déplacement du bateau espion ou bien du drone, toutes les torpilles en jeu se déplacent de manière orthogonale dans la grille. Chaque torpille est indépendante, ce qui signifie que plusieurs torpilles peuvent se déplacer dans des directions différentes au même moment. L'*intelligence* est responsable de déterminer la direction des déplacements pour chaque torpille.

   Dans l'exemple ci-bas, le bateau espion lance un missile sur l'avant du bateau bleu, ce qui déclenche une torpille. L'*intelligence* positionne la torpille dans le coin du périmètre. Le bateau espion se déplace ensuite vers l'ouest, permettant à l'*intelligence* d'avancer la torpille vers le nord.



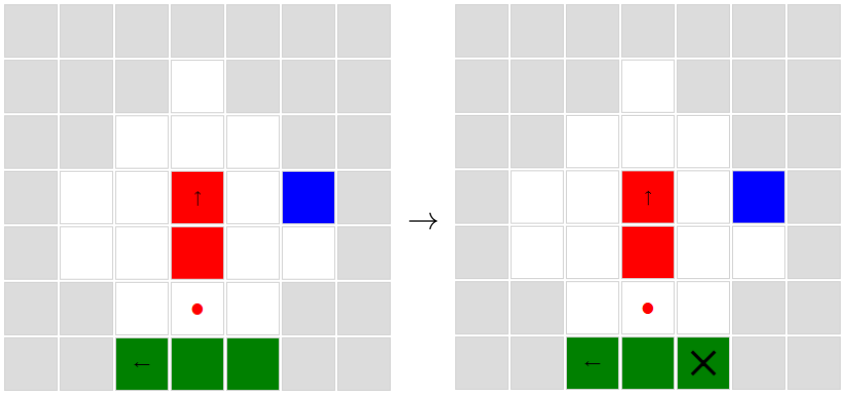
   Si une torpille se trouve dans la zone dégagée par le radar du bateau espion, le joueur peut détruire cette torpille en lançant un missile à sa position. Dès que la torpille atteint le bateau espion, il perd un point de vie pour la cellule touchée.

   Une fois déployée, une torpille demeure sur la grille tant qu'elle n'a pas rencontré un bateau ou qu'elle n'est pas détruite par le joueur, même si le bateau bleu est coulé entre-temps.

7. **Cruiser** : Tant qu'il n'est pas coulé, le bateau vert peut rediriger un missile vers lui-même et encaisser le choc à la place de la torpille ou du bateau initialement visé par le joueur. L'*intelligence* pourrait préférer de ne pas rediriger de missiles et les laisser atteindre leur cible normalement.

   Dans la situation ci-bas, le joueur souhaite lancer un missile sur la seule cellule accessible du bateau bleu. L'*intelligence* choisi de rediriger le missile vers la dernière

cellule occupée par le bateau vert. Notez que le bateau bleu n'a pas été endommagé, donc aucune torpille n'est déployée dans cette situation. De plus, la cellule sélectionnée pour encaisser le choc ne doit pas nécessairement être dégagée par le radar du bateau espion.



Il est possible de volontairement couler le bateau vert pour sauver une torpille ou épargner un point de vie à un bateau plus important selon le contexte, toujours en fonction de l'expertise de l'*intelligence* et de la stratégie priorisée.

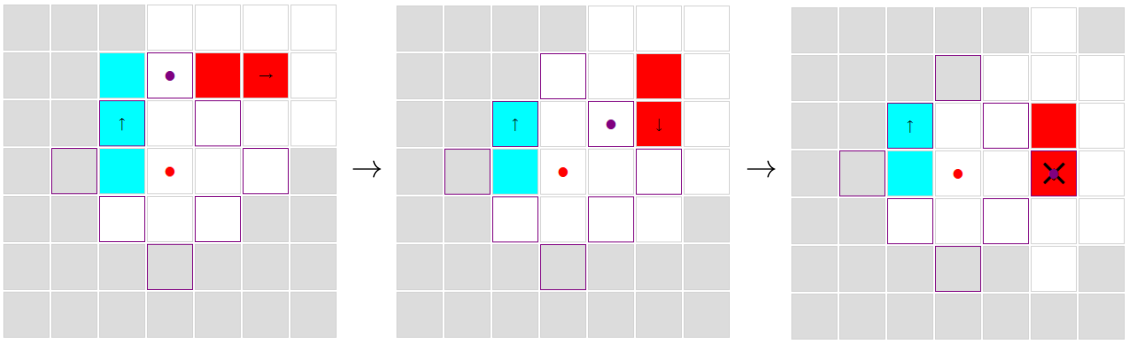
8. **AircraftCarrier** : Le bateau violet sert de centre de contrôle pour l'avion. Ce bateau est responsable des communications avec l'avion pour orchestrer les déplacements et la modification de l'itinéraire de l'avion, jusqu'au moment ou le bateau violet est coulé. À chaque fois que l'avion survole le bateau espion, il lance directement un missile sur ce dernier, endommageant ainsi la cellule sous l'avion.

   Dès le début de la partie, l'itinéraire de l'avion est indiqué sur la grille de jeu via des bordures violettes autour des cellules formant l'itinéraire. Cet itinéraire est déterminé par nul autre que l'*intelligence*. En tout temps, l'itinéraire doit entièrement être compris sur la grille de jeu, sans quoi l'avion risque de quitter la grille. L'itinéraire est en réalité un flot infini de coordonnées.

   L'itinéraire doit s'adapter aux dimensions de la grille. Vous pouvez assumer que la taille minimale est de 5 rangées et 5 colonnes, puisque le bateau violet de taille 5 est présent sur la grille lorsque vient le temps de tracer l'itinéraire.

   L'avion n'est évidemment pas soumis au trafic maritime et il peut cohabiter avec le drone sur la même cellule. Il se déplace donc librement en parcourant son itinéraire de manière cyclique. L'avion est la seule entité qui peut se déplacer en diagonal si son itinéraire le permet. Après avoir survolé le bateau espion, l'avion continue normalement son vol en fonction de l'itinéraire prédéterminé.

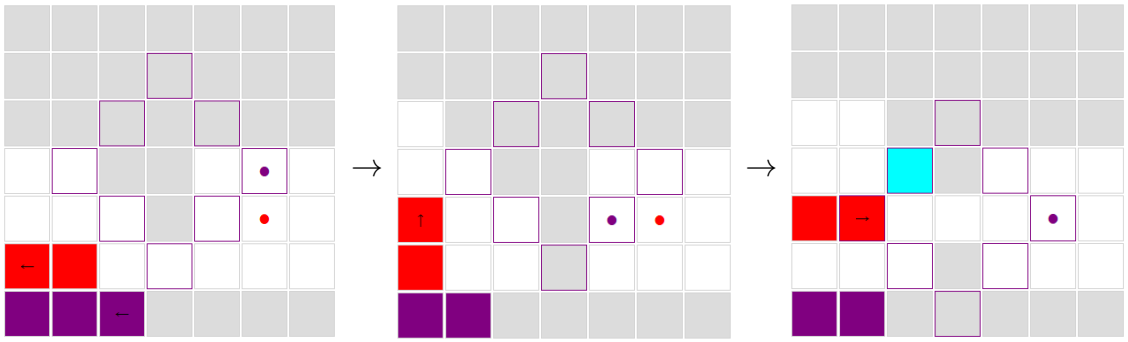
   On représente l'avion par un point violet dans la grille. L'avion se situe initialement dans la première cellule de l'itinéraire. Après chaque déplacement du bateau ou bien du drone, l'avion avance obligatoirement dans la cellule suivante de son itinéraire.



Dans la configuration ci-haut, l'avion se déplace dans le sens horaire selon ce qui a été choisi par l'*intelligence* lors de la création de l'itinéraire. Après deux déplacements du bateau espion, l'avion se retrouve maintenant directement au-dessus de la tête du bateau espion. Ce dernier perd instantanément le point de vie associé à la cellule sous l'avion.

Dès que le bateau espion est repéré dans le périmètre d'un bateau adverse autre que le bateau lime ou violet, l'*intelligence* peut décaler l'itinéraire dans une direction cardinale. Cette capacité spéciale est vérifiée après chaque déplacement du bateau espion ou du drone. Par exemple, l'itinéraire pourrait être décalé un maximum de n fois si le bateau espion navigue dans le périmètre des bateaux aqua, bleu ou vert pendant n tours de suite.

Finalement, l'*intelligence* a également la possibilité d'inverser le sens de l'itinéraire lorsque le bateau espion entre dans le périmètre du bateau violet. Encore une fois, la vérification est lancée après chaque déplacement du bateau espion ou du drone. L'itinéraire pourrait alors être inversé un maximum de n fois pour n tours consécutifs où le bateau espion est repéré dans le périmètre du bateau violet.



Dans l'exemple ci haut, l'avion parcourt son itinéraire dans le sens horaire. Le bateau se déplace en provoquant une progression de l'avion sur son itinéraire. Après le premier déplacement, le bateau espion est encore dans le périmètre du bateau violet. L'*intelligence* décide alors d'inverser l'itinéraire de l'avion. Au prochain déplacement, l'avion avance maintenant dans le sens antihoraire. Le bateau espion se trouve cependant dans le périmètre du bateau aqua, ce qui déclenche un décalage de l'itinéraire vers le sud selon la stratégie adoptée par l'*intelligence*.

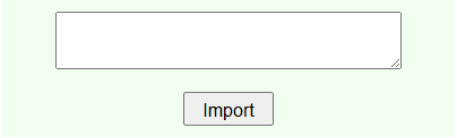
9. **Fin de la partie** : L'objectif du joueur est de couler tous les bateaux adverses en lançant des missiles. Une fois qu'un bateau est coulé, sa capacité spéciale est

désactivée pour le reste de la partie, à l'exception des torpilles existantes qui demeurent en jeu.

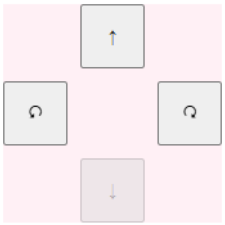
L'objectif de l'*intelligence* est de couler le bateau espion en exploitant les capacités spéciales des bateaux disponibles et de ses connaissances concernant l'état de la grille. La notion de brouillard ne s'applique pas à l'*intelligence*.

La partie prend immédiatement fin lorsqu'un des deux camps remplit son objectif.

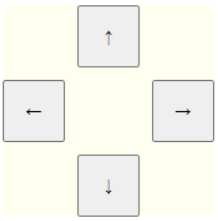
10. **Contrôles** : Pour lancer une partie, l'utilisateur doit importer une configuration préalablement créée dans la page `Fleet Deployment` de l'application. Une zone de texte est prête à accueillir les données au format JSON. Les autres contrôles sont ensuite déverrouillés une fois que les données sont chargées sur la grille.



Le joueur peut ensuite naviguer le bateau espion dans la grille à l'aide des boutons directionnels. Les déplacements du bateau espion imitent le comportement naturel d'un bateau souhaitant naviguer à travers un champ de bataille.



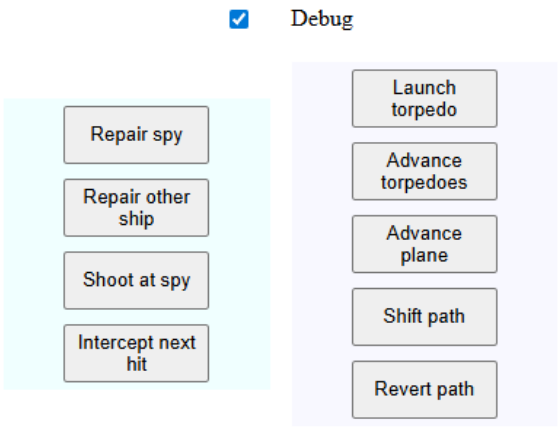
Un second ensemble de boutons directionnels permettent de déplacer le drone dans la grille de jeu. Le drone peut se déplacer dans n'importe quelle direction à tout moment, à condition de ne pas quitter la grille.



Le joueur peut finalement cliquer sur la grille pour lancer des missiles vers des bateaux adverses ou encore des torpilles s'approchant dangereusement du bateau espion.

Pour tester les implémentations et les stratégies développées pour l'*intelligence*, un mode debug est offert par défaut en lançant une partie. Lorsque le mode debug est activé, aucune capacité spéciale des bateaux adverses n'est déclenchée suite à une

action du joueur. De plus, la détection de fin de partie n'est pas considérée tant que le mode debug est actif.



Le panneau debug présente également quelques boutons qui permettent d'activer manuellement la plupart des capacités spéciales sur demande. Les fonctions appelées par ces boutons sont les mêmes que celles normalement appelées pendant le déroulement standard du jeu.

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 nécessaire 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 toute 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. **Modifications.**

   Pour amorcer cette deuxième partie du projet, vous devez modifier le type suivant :

```
type Sector = Clear | Active of Name * int * bool | Torpedo
```

Un secteur peut désormais être occupé par une torpille ou bien un bateau possiblement endommagé. Le troisième élément d'une cellule occupée par un bateau est utilisé pour déterminer les points de vie restant d'un bateau. La valeur `false` signifie que le point de vie est encore disponible, alors que la valeur `true` déclare que le bateau a été touché à cet endroit.

Vous devez corriger toutes les fonctions de la première partie du projet pour intégrer cette modification afin d'être en mesure de compiler le programme à nouveau. Le comportement des fonctions demeure inchangé, à l'exception de la fonction `Battlefield.replaceShip` qui est expliquée plus bas. Les torpilles et les points de vie des bateaux ne sont pas utilisés par ces fonctions. Vous pouvez lancer le programme de la première partie du projet pour valider que tout fonctionne correctement avec le nouveau type.

On vous demande par la suite de copier vos fonctions dans le gabarit partagé pour la deuxième partie du travail. Vous trouverez dans ce gabarit de nouvelles fonctions à implémenter ainsi que deux nouveaux modules à remplir.

2. **Fonctions.**

La prochaine étape est de compléter les fonctions qui vous sont demandées. Pour ce faire, vous êtes limités à la partie "fonctionnelle" du langage *F#*. On vous permet également l'utilisation des références et des variables mutables pour cette deuxième partie du projet, mais les boucles traditionnelles demeurent interdites. Vous avez aussi maintenant l'autorisation de changer l'ordre des fonctions à l'intérieur d'un module afin de faciliter l'implémentation des fonctions et encourager la réutilisation du code.

La documentation plus haut présente les interactions possibles entre l'interface utilisateur et la librairie *F#*. Les fonctions à compléter sont réparties dans 4 modules :

- `Navigation` :
  - *canMoveDrone* (*drone* : *Coord*) (*direction* : *Direction*) (*grid* : *Sector Grid*) : *bool*
    Vérifie s'il est possible de déplacer le drone dans la direction spécifiée. Le drone peut se déplacer librement sur la grille, à condition qu'il ne tente pas de sortir de la grille.

  - *moveDrone* (*drone* : *Coord*) (*direction* : *Direction*) : *Coord*
    Déplace le drone dans la direction spécifiée.

- `Battlefield` :
  - *replaceShip* (*ship* : *Ship*) (*grid* : *Sector Grid*) : *Sector Grid*
    Produit une nouvelle grille où le bateau donné est repositionné dans la grille. Le bateau doit conserver le même nombre de points de vie, où chaque cellule touchée est dans le même ordre relatif selon ses coordonnées et sa direction. De plus, si le bateau est maintenant placé sur une torpille, celle-ci explose suite à

la collision et le bateau perd un point de vie pour cette cellule.

- *getFog* (*grid*: *Sector Grid*) (*drone*: *Coord*) : *bool Grid*
  Produit une grille de booléens représentant le brouillard. Les cellules ont une valeur `false` par défaut. Seules les cellules comprises dans le radar du bateau ou du drone sont assignées une valeur `true`.

- *isRevealed* (*coord*: *Coord*) (*fog*: *bool Grid*) : *bool*
  Détermine si une certaine cellule est dégagée du brouillard ou non. Utilisée pour l'affichage graphique du brouillard seulement.

- *getTorpedoes* (*grid*: *Sector Grid*) : *Coord list*
  Retourne une liste de coordonnées où chaque torpille est présentement positionnée sur la grille. Si la grille ne contient aucune torpille, alors la liste de retour est vide.

- *isHit* (*coord*: *Coord*) (*grid*: *Sector Grid*) : *bool*
  Détermine si une certaine cellule représente un secteur actif et endommagé pour un bateau quelconque. Utilisée principalement pour l'affichage graphique des points de vie.

- *canHit* (*coord*: *Coord*) (*grid*: *Sector Grid*) : *bool*
  Détermine si le joueur peut lancer un missile sur la cellule visée. Les seules cellules admissibles sont celles à l'intérieur du radar du bateau espion qui contiennent soit une torpille, soit un bateau adverse qui n'est pas déjà endommagé à cet endroit. Utilisée lorsque le joueur clique sur la grille.

- *hit* (*coord*: *Coord*) (*grid*: *Sector Grid*) : *Sector Grid*
  Produit une nouvelle grille où le bateau visé a été touché sur la cellule spécifiée.

- *getRemainingHealth* (*name*: *Name*) (*grid*: *Sector Grid*) : *int*
  Retourne le nombre de points de vie restant d'un bateau à partir de sa classe. Le nombre de points de vie maximal est la taille du bateau, tandis que le nombre minimal est 0 si le bateau est coulé. Si le bateau de la classe demandée n'apparaît pas sur la grille, on considère qu'il lui reste 0 points de vie. Utilisée par l'interface utilisateur pour la détection des bateaux coulés et pour déterminer si les capacités spéciales des bateaux s'appliquent.

- `Intelligence` :
  - *repairOtherShip* (*grid*: *Sector Grid*) : *Sector Grid*
    Capacité spéciale du bateau lime. Redonne un point de vie à un bateau adverse si c'est possible de le faire.

  - *repairSpy* (*grid*: *Sector Grid*) : *Sector Grid*
    Utilisée lorsque le bateau espion parvient à couler le bateau lime. Redonne un point de vie au bateau espion.

  - *shootAtSpy* (*grid*: *Sector Grid*) : *Sector Grid*
    Capacité spéciale du bateau aqua. Lance un missile sur une cellule du bateau

espion au moment où le bateau aqua est coulé.

- *launchTorpedo* (*grid*: *Sector Grid*) : *Sector Grid*
  Capacité spéciale du bateau bleu. Ajoute une torpille dans la grille. La torpille doit être positionnée sur une cellule faisant partie du périmètre du bateau bleu.

- *advanceTorpedoes* (*grid*: *Sector Grid*) : *Sector Grid*
  Appelée à chaque déplacement du bateau espion ou du drone. Toutes les torpilles présentes dans la grille se déplacent obligatoirement dans une direction cardinale. Si une torpille rencontre un bateau, ce dernier perd le point de vie associé à la cellule où le contact a eu lieu.

- *interceptNextHit* (*coord*: *Coord*) (*grid*: *Sector Grid*) : *Sector Grid*
  Capacité spéciale du bateau vert. Retourne une nouvelle grille où le bateau vert a potentiellement rediriger le missile vers lui-même pour absorber l'impact. Si tel est le cas, alors le bateau vert doit perdre un point de vie. Sinon, la cellule visée est touché normalement.

- *getPlanePath* (*grid*: *Sector Grid*) : *Coord list*
  Crée un liste de coordonnées qui représente l'itinéraire de l'avion en respectant les dimensions de la grille. La liste doit être continue, ce qui signifie que deux coordonnées consécutives doivent soit partager la même rangée ou la même colonne, soit être placées directement en diagonal. Une coordonnée peut apparaître plus d'une fois dans l'itinéraire.

- *advancePlane* (*stream*: *Coord Stream*) : *Coord Stream*
  Appelée à chaque déplacement du bateau espion ou du drone. Fait progresser le flot infini de coordonnées. Le flot résultant débute donc au second élément du flot reçu en paramètre.

- *shiftPath* (*grid*: *Sector Grid*) (*stream*: *Coord Stream*) : *Coord Stream*
  Appelée après chaque déplacement du bateau espion ou du drone si le bateau espion se trouve dans le périmètre d'un bateau adverse aqua, bleu ou vert. Décale potentiellement l'itinéraire de l'avion d'une seule position dans une direction cardinale. Si aucun décalage n'est appliqué, le flot original est retourné.

- *revertPath* (*grid*: *Sector Grid*) (*stream*: *Coord Stream*) : *Coord Stream*
  Appelée après chaque déplacement du bateau espion ou du drone si le bateau espion se trouve dans le périmètre du bateau violet. Inverse potentiellement l'itinéraire de l'avion dans le sens opposé, sans avancer l'avion. Si l'inversion n'a pas lieu, le flot original est retourné.

- `Stream` :
  - *cycleList* (*l* :′ *a list*) :′ *a Stream*
    Crée un flot infini en parcourant la liste de manière cyclique à partir du premier élément de la liste.

3. **Intelligence.**

   Pour chaque fonction à compléter dans le module `Intelligence`, présentez brièvement la stratégie retenue dans votre rapport. Les stratégies n'ont pas besoin d'être très sophistiquées. Plus votre *intelligence* est puissante, plus il sera difficile de gagner une partie !

   Ajoutez une petite discussions sur les forces et faiblesses de vos stratégies. Expliquez votre cheminement pour les identifier, puis mentionnez quelques améliorations possibles.

   Les stratégies aléatoires ou qui ignorent complètement l'état de la grille ne sont pas permises.

4. **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. N'oubliez pas que vous pouvez définir des fonctions d'ordre supérieur pour regrouper des comportements communs qui ne dépendent pas du type des valeurs manipulées. 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.

   Pour ce deuxième travail, on vous demande seulement d'intégrer dans votre rapport une brève discussion à propos 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.

5. **Vidéo.**

   Une fois que votre travail est terminé, enregistrez une courte vidéo d'une durée maximale de 3 minutes (180 secondes) 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. Voici les fonctionnalités à inclure dans votre démonstration, sans ordre précis :

   - Lancement d'une partie (*Play*) :
     - Importer une configuration précédemment créée.

- Lancer des missiles sur des bateaux adverses.
- Montrer l'interception d'un missile par le bateau vert.
- Montrer qu'un point de vie a été redonné à un bateau adverse par le bateau lime.
- Perdre un point de vie sur le bateau espion en coulant le bateau aqua.
- Ajouter une torpille à l'aide du bateau bleu.
- Montrer le déplacement des torpilles.
- Détruire une torpille en lançant un missile.
- Perdre un point de vie sur le bateau espion suite à une collision avec une torpille.
- Couler le bateau lime pour récupérer un point de vie.
- Déplacer l'itinéraire de l'avion.
- Inverser l'itinéraire de l'avion.
- Perdre un point de vie sur le bateau espion en étant positionné sous l'avion.

Vous aurez certainement besoin du mode debug pour visiter l'ensemble de ces fonctionnalités. Plusieurs points sont interchangeables. Assurez-vous seulement de bien montrer chaque capacité spéciale, sans vous soucier des multiples combinaisons d'actions et situations complexes possibles.

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

6. **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, quelques explications et discussions à propos des stratégies choisies pour le développement de votre *intelligence* ainsi que certaines observations en lien avec les 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
Intelligence	-1	1	Stratégies aléatoires ou qui ignorent l'état du jeu	-2
			Effort insuffisant	-1
Abstraction	0	2	Effort insuffisant	-2
			Les fonctions définies ne sont pas réutilisées	-1
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 (`bin`, `obj`, `.vs`, `.idea`) inclus dans la remise	-2
			Fichier `README.md` absent ou non complété	-1