École de technologie supérieure **Auteur :** Iannick Gagnon

INF136 Travail Pratique No. 2

Bataille Navale Travail en équipe de 4

1. 

1 Instructions

- Vous devez remettre une seule archive qui contient les fichiers suivants :

    1. `constantes.py`
    2. `flotte.py`
    3. `grille.py`
    4. `jeu.py`
    5. `main.py`
    6. `navire.py`

- La **fonctionnalité**, la **lisibilité**, la **documentation** et **l'efficacité** du code seront évaluées suivant le gabarit de correction qui vous a été fourni.

2 Objectifs

- Utiliser l'ensemble des notions vues jusqu'à présent avec une emphase sur les **collections** et les **tableaux**.

3 Contraintes

- L'usage de variables globales entraîne une pénalité de 25%.

4 Mise en situation

Vous devez implémenter un simulateur du jeu de société Bataille Navale (*Battleship*) qui place deux joueurs l'un contre l'autre sur une grille qui représente un champ de bataille maritime :

Figure 1 – Exemple d'une vraie planche de jeu Bataille Navale

5 Description de l'application

Vous devez réaliser une application Python qui simule le jeu de société Bataille Naval. L'application compte 6 modules dont 2 qui sont fournis et identifiés en vert :

2. `constantes.py`
3. `navire.py`

Voici la structure du programme à développer :

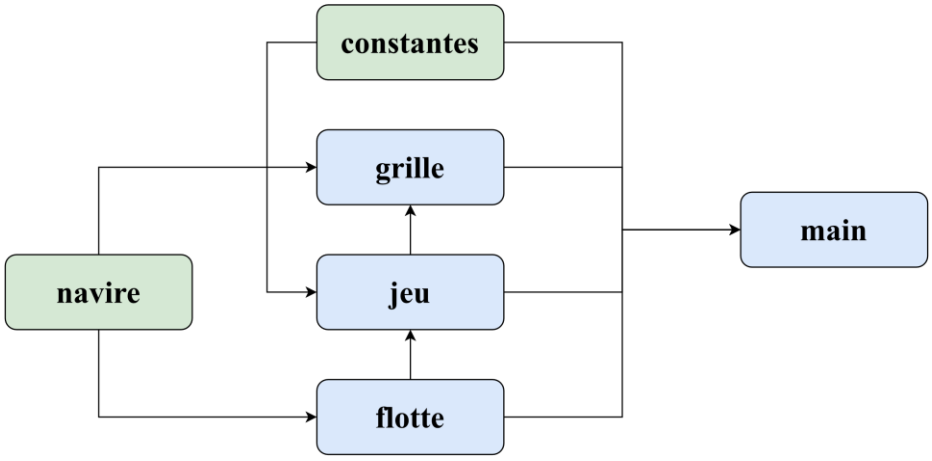


Figure 2 – Structure de l'application

Voici une description sommaire de chacun des modules à compléter :

Nom	Description
main	Procédure de démarrage du jeu.
grille	Sous-programmes qui permettent de créer et manipuler la grille de jeu.
flotte	Sous-programmes gérer une flotte de navires.
jeu	Sous-programmes qui permettent de gérer une séance de jeux.

Voici une description sommaire des modules fournis que **vous ne devez pas modifier** :

Nom	Description
constantes	Constantes du jeu. **Prenez-en connaissance dès le début du projet.**
navire	Encapsulation d'un navire dans un dictionnaire.

6 Test unitaires

Une suite de tests unitaires vous est fournie pour les modules `flotte`, `jeu` et `grille`. Ces tests seront utilisés pour la correction, mais leur **succès ne garantit pas le bon fonctionnement de votre programme**. Ils sont offerts à titre d'aide au développement.

7 Dépendances externes

Vous devez installer la librairie `deepdiff` qui est utilisée dans `test_grille.py` ainsi que la librairie `pytest` pour exécuter les tests fournis dans `test_flotte.py`, `test_grille.py` et `test_jeu.py`.

8 Ordre de développement

Le schéma de la Figure 2 impose l'ordre de développement suivant :

`flotte → jeu → grille → main`

9 Les flottes de navires (`navire.py`)

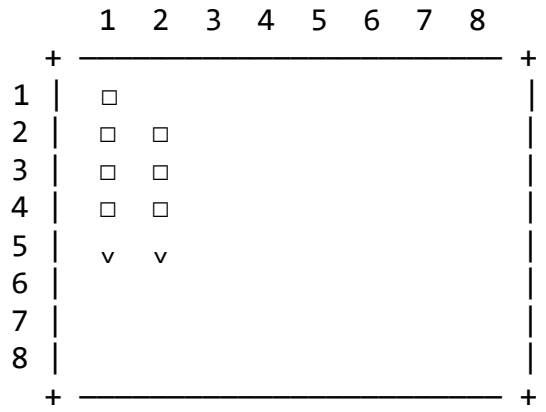
Une flotte de navires qui est en fait **une liste qui contient des dictionnaires**. Les dictionnaires **encapsulent** les navires sous la forme d'**enregistrements** avec les **champs** suivants :

Champ	Type	Description
`'taille'`	`int`	Nombre total de cases occupées par le navire.
`'vie'`	`int`	Nombre de cases qui n'ont pas été atteintes par l'adversaire.
`'i'`	`int`	Coordonnée `i` de la tête du navire.
`'j'`	`int`	Coordonnée `j` de la tête du navire.
`'orientation'`	`int`	Un entier qui encode l'orientation du navire
`'positions'`	`list`	Les coordonnées des parties du navire qui n'ont pas été atteintes par l'adversaire.

Voici quelques précisions qui doivent être comprises pour bien visualiser la mécanique du jeu :

- Au départ, les champs `taille` et `vie` sont égaux, car le navire est intact.
- La `vie` correspond au nombre d'éléments du champ `positions` qui est une liste.
- Les `positions` sont stockées dans une **liste de tuples** qui contiennent les coordonnées `(i,j)` des parties du intactes du navire.
- Un navire coule lorsque sa `vie` devient nulle ou, de manière équivalente, que la liste de `positions` devient vide.
- L'`orientation` d'un navire est encodée par un entier qui correspond aux constantes nommées `NORD`, `SUD`, `EST` et `OUEST` de `constantes.py`.
- Si l'on connaît la position de la tête du navire, soit `i = 1` et `j = 2` par exemple, que l'on connaît son `orientation`, soit le Nord, et finalement que l'on connaît aussi sa `taille`, disons 3, il est possible de déduire ces `positions` initiales, soient `[(1,2), (2,2), (3,2)]`.

Le schéma suivant représente une grille de jeu qui contient 2 navires dont les têtes sont identifiées par les bouts de flèches et le corps par des carrés (□):



Si vous avez compris les explications de la page précédente, vous êtes en mesure de **déduire** la forme que prend leurs enregistrements, soient :

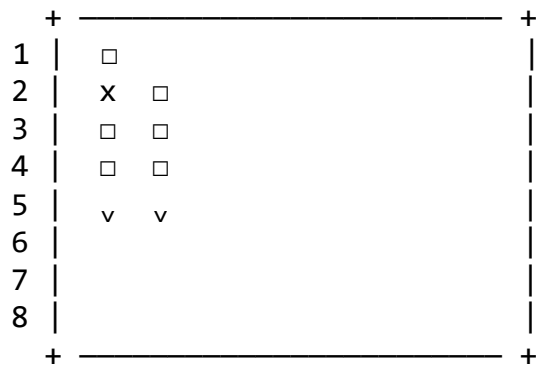
taille	5
vie	5
i	5
j	1
orientation	1
positions	[(5,1), (4,1), (3,1), (2,1), (1,1)]

taille	4
vie	4
i	5
j	2
orientation	1
positions	[(5,2), (4,2), (3,2), (2,2)]

Au départ, la `taille` et la `vie` correspondent au nombre de cases occupées par chacun des navires. Les coordonnées `i` et `j` correspondent aux positions des têtes des navires, soient `(5,1)` et `(5,2)`. L'`orientation` est la même pour les deux, soit le Sud, qui correspond à la constante nommée `SUD` qui vaut 1. Les `positions` découlent logiquement des autres propriétés du navire.

Lorsqu'un navire est touché, un `X` est inséré à la position atteinte comme le montre le schéma suivant :

1 2 3 4 5 6 7 8



Le premier navire correspond maintenant à l'enregistrement suivant :

taille	5
vie	4
i	5
j	1
orientation	1
positions	[(5,1), (4,1), (3,1), ~~(2,1)~~, (1,1)]

Vous devez compléter le Quiz du TP2 sur Moodle avant de continuer.

i. SP1 `flotte_genererer_aleatoire`

Description :

Génère une flotte de navires aléatoires.

Paramètres : Aucuns.

Valeur de retour : Une flotte de navires aléatoires. [`list`]

Requis :

1. La taille de la flotte est un nombre aléatoire situé entre `FLOTTE_TAILLE_MIN` et `FLOTTE_TAILLE_MAX` avec la fonction `randint` de la librairie `random`.
2. Les navires sont générés à l'aide de la fonction `navire_generer_aleatoire()` du module `navire`.

ii. SP2 `flotte_calculer_vie`

Description :

Calcule la `'vie'` d'un joueur à partir de sa flotte. Celle-ci correspond au nombre de tirs qu'il ou qu'elle peut encaisser avant de perdre la partie.

Paramètres : `flotte`: Une flotte de navires. [`list`]

Valeur de retour : Somme des champs `'vie'` des navires de la flotte. [`int`]

Requis :

3. Accéder aux champs `'vie'` des navires avec l'accesseur `navire_obtenir_vie()`.

iii. SP3 `flotte_gestionnaire_tir`

Description :

Met une flotte à jour par suite d'un tir à une position donnée. Si un navire est atteint, la position du tir est supprimée de la liste des positions qu'il occupe et sa vie est décrémentée d'une unité.

Paramètres :

- `flotte`: Une flotte de navires. [`list`]
- `i_tir`: La ligne du tir [int]
- `j_tir`: La colonne du tir. [`int`]

Valeur de retour : Indicateur de succès du tir à atteindre une cible. [`bool`]

Requis :

4. Parcourir les positions de chacun des navires de la flotte à la recherche d'un touché.

5. Les positions d'un navire correspondent à une liste de tuples `(i,j)` actifs d'un navire retournés par l'accesseur `navire_obtenir_positions()`.

6. Si l'une des positions du navire correspond à la position du tir, il faut la retirer de sa liste de positions et décrémenter la vie du navire avec la fonction `navire_decrementer_vie()`.

7. Si un navire est atteint, retourner Vrai, sinon Faux.

10 Services de gestion du jeu (jeu.py)

Cette section décrit les sous-programmes utilisés pour la gestion d'une partie.

iv. SP4 jeu_afficher_joueur_courant

Description :

Affiche un message qui indique le joueur courant.

Paramètres : `identifiant`: L'identifiant du joueur courant. [`int`]

Valeur de retour : Aucune.

Requis :

8. Si c'est au tour du joueur (`JOUEUR`), afficher le message suivant :

```
C'est au tour du joueur
```

9. Si c'est au tour de l'ordinateur (`ORDINATEUR`), afficher le message suivant :

```
C'est au tour de l'ordinateur
```

v. SP5 jeu_joueur_saisir_position_tir

Description :

Saisit la position du tir du joueur.

Paramètres : Aucuns.

Valeur de retour : Position `(i,j)` du tir. [`tuple`]

Requis :

10. Faire la saisie jusqu'à ce que la position soit valide en utilisant les dimensions de la grille qui correspondent aux constantes `HAUTEUR_GRILLE` et `LARGEUR_GRILLE`.

vi. SP6 `jeu_ordinateur_saisir_position_tir`

Description :

Saisit la position du tir de l'ordinateur.

Paramètres : Aucuns.

Valeur de retour : Position `(i,j)` du tir. [`tuple`]

Requis :

1. Utiliser la fonction `randint` de la librairie `random`.
2. Générer une position aléatoire tant qu'il ne s'agit pas d'une cellule vide.

vii. SP7 `jeu_est_partie_terminee`

Description :

Détermine si une partie est terminée si la vie d'une des flottes est égale à zéro.

Paramètres :

- `flotte_1`: Une flotte de navires du premier joueur. [`list`]
- `flotte_2`: Une flotte de navires du deuxième joueur. [`list`]

Valeur de retour : Indicateur de fin de partie. [`bool`]

Requis :

1. Calculer la vie des flottes avec la fonction `flotte_calculer_vie()`.
2. Retourner Vrai si la partie est terminée, sinon Faux.

viii. SP8 `jeu_afficher_vainqueur`

Description :

Annonce le vainqueur de la partie.

Paramètres :

- `flotte_joueur`: La flotte de navires du joueur. [`list`]

Valeur de retour : Aucune.

Requis :

1. Si c'est l'ordinateur qui a gagné, afficher le message suivant :

```
C'est l'ordinateur qui a gagné!
```

2. Si c'est le joueur qui a gagné, afficher le message suivant :

```
C'est le joueur qui a gagné!
```

11 Gestion de la grille de jeu (`grille.py`)

Cette section décrit les sous-programmes nécessaires pour la gestion d'une grille de jeu. **Elle fait référence à une série de constantes avec lesquelles vous devez vous familiariser.** Ces constantes représentent des codes qui correspondent aux états possibles des cellules de la grille et leurs symboles correspondants. Les codes sont associés à leurs symboles dans la constante `SYMBOLES_GRILLE` qui est un **dictionnaire** de la forme `constante:symbole`.

ix. SP9 `grille_initialiser`

Description :

Initialise une grille vierge aux dimensions données.

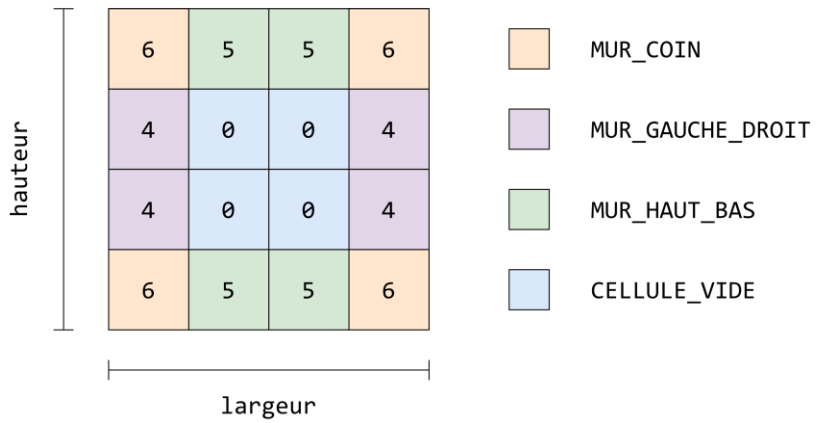
Paramètres :

- `hauteur`: Nombre de cases en hauteur de la grille. [`int`]
- `largeur`: Nombre de cases en largeur de la grille. [`int`]

Valeur de retour : Grille de jeu vierge. [`list`]

Requis :

1. Initialiser et retourner un tableau 2D rempli avec les **constantes symboliques** suivantes définies dans `grille.py` :



Une grille 4×4 est donc initialisée à : `[[6,5,5,6],[4,0,0,4],[4,0,0,4],[6,5,5,6]]`.

x. SP10 `grille_dessiner`

Description :

Crée une grille de symboles [`str`] à partir de la grille de codes [`int`] (SP9) pour l'affichage.

Paramètres :

- `grille`: Grille de jeu qui contient des constantes symboliques (voir SP9). [`list`]

Valeur de retour : Une grille contenant des symboles. [`list`]

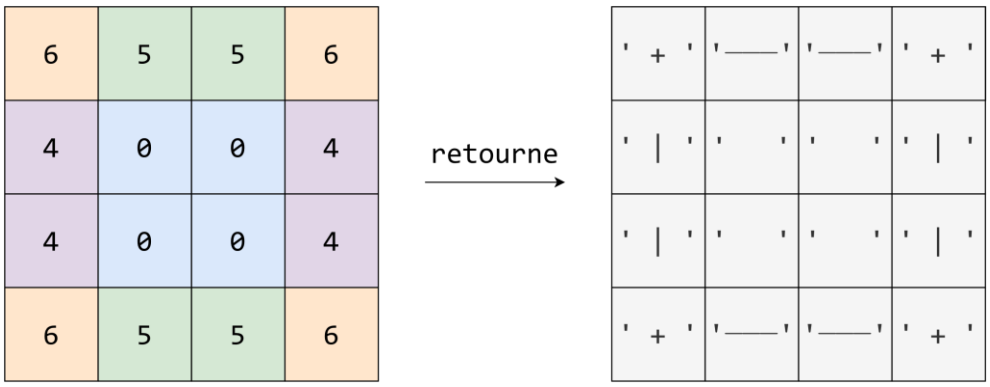
Requis :

2. Créer une nouvelle grille.

3. Convertir les constantes symboliques de la grille d'origine en symboles en utilisant la constante `SYMBOLES_GRILLE` qui est un dictionnaire qui possèdes les constantes symboliques comme clés et les symboles correspondants comme valeurs. Par exemple :

```
>> SYMBOLES_GRILLE[MUR_COIN]
 +

>> SYMBOLES_GRILLE[MUR_HAUT_BAS]
---
```



xi. SP11 `grille_afficher`

Description :

Affiche une grille de symboles à la ligne de commande avec les numéros de lignes et de colonnes en partant de 1.

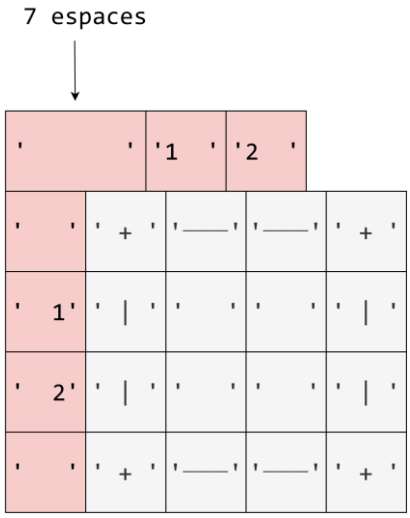
Paramètres :

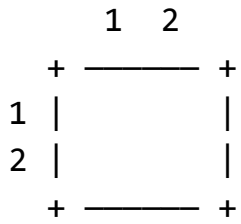
- `grille`: Grille de jeu qui contient des symboles (voir SP10). [`list`]

Valeur de retour : Aucune.

Requis :

4. Pour montrer les numéros de lignes et de colonnes, prenant l'exemple du SP10, il faut ajouter les éléments suivants en couleur rouge dans le schéma au moment de l'affichage (c.-à-d. faites-le dans le `print` sans modifier la grille) :





En mots, la première ligne est constituée d'un premier bloc de 7 espaces vides suivi d'autant de blocs de 3 espaces de largeur contenant le numéro de colonne **aligné à gauche** (voir schéma ci-dessus) qu'il y a de colonnes. Les lignes subséquentes commencent par soit 3 espaces vides, soit un bloc de trois espaces de largeur avec le numéro de ligne **aligné à droite** (voir schéma ci-dessus).

xii. SP12 `grille_generer_position_aleatoire`

Description :

Génère une position aléatoire dans une grille **en évitant les bordures**.

Paramètres :

- `grille`: Grille de jeu. [`list`]

Valeur de retour : Une position `(i,j)` aléatoire. [`tuple`]

Requis :

5. Utiliser la fonction `randint` de la librairie `random`.
6. Il faut empêcher la fonction de générer des positions situées sur les bordures incluant les coins. La grille 4×4 des exemples précédents ne possède donc que 4 cases valides.

xiii. SP13 `grille_cellule_est_vide`

Description :

Indique si la cellule ciblée est vide.

Paramètres :

- `grille`: Grille de jeu. [`list`]
- `i`: Ligne ciblée. [`int`]
- `j`: Colonne ciblée. [`int`]

Valeur de retour : Indicateur de vide. [`bool`]

Requis :

7. Comparer le contenu de la cellule avec la constante `CELLULE_VIDE`.

xiv. SP14 `grille_generer_position_aleatoire_valide`

Description :

Génère une position aléatoire dans une grille qui est à l'intérieur des bordures et **qui ne contient pas déjà un navire**.

Paramètres :

- `grille`: Grille de jeu. [`list`]
- `max_iter`: Le nombre maximal d'itérations permis **fixé à 50 par défaut**. [`int`]

Valeur de retour : Position `(i,j)` aléatoire valide. [`tuple`]

Requis :

8. Générer des positions aléatoires avec `grille_generer_position_aleatoire()` **tant et aussi longtemps qu**e celle-ci n'est pas vide avec `grille_cellule_est_vide()` et que le nombre d'itérations ne dépasse pas la limite égale à `max_iter`.

9. En cas d'échec, déclencher une erreur d'**assertion** avec le message suivant :

```
Nombre d'itérations maximal atteint.
```

xv. SP15 `grille_navire_est_valide`

Description :

Indique si la position d'un navire est valide en fonction de l'état actuel de la grille, de l'orientation du navire et de sa taille.

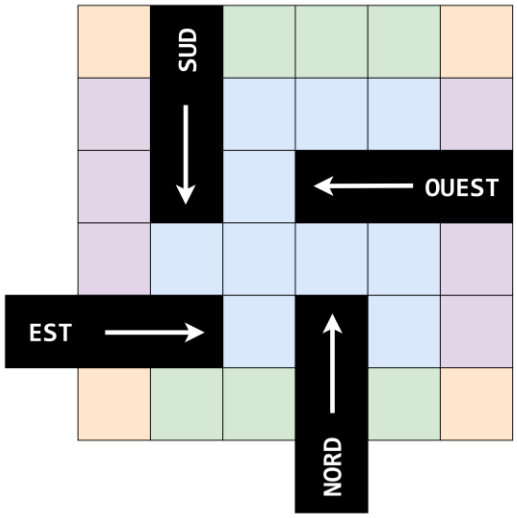
Paramètres :

- `grille`: Grille de jeu. [`list`]
- `i`: Ligne de la position de la tête du navire. [`int`]
- `j`: Colonne de la position de la tête du navire. [`int`]
- `orientation`: Orientation du navire. [`int`]
- `taille`: Nombre de cases occupées par le navire. [`int`]

Valeur de retour : Indicateur de validité. [`bool`]

Requis :

1. **Vérifier que le navire ne déborde pas de la grille.** Pour cela, commencez par la tête du navire et développez un mécanisme qui permet de déterminer si toutes les cases impliquées par son orientation (constantes `NORD`, `SUD`, `EST`, `OUEST`) et sa taille se trouvent à l'intérieur des limites (c.-à-d. les cases au centre en bleu). Voici des exemples de navires de taille 3 qui ne sont pas valides avec des flèches qui pointent vers leurs têtes :



2. **Vérifier qu'il n'y a pas de chevauchements entre navires.** Pour cela, assurez-vous que toutes les cellules impliquées par la position de la tête, l'orientation et la taille du navire sont vides avec `grille_cellule_est_vide()`.

3. Si tout est bon, on dit que le navire est valide et on retourne Vrai, sinon Faux.

xvi. SP16 `grille_generer_orientation_aleatoire_valide`

Description :

Génère une orientation aléatoire pour un navire en tenant compte de l'état d'une grille donnée de manière à éviter les chevauchements de navires et des débordements de la grille.

Paramètres :

- `grille`: Grille de jeu. [`list`]
- `i`: Ligne de la position de la tête du navire. [`int`]
- `j`: Colonne de la position de la tête du navire. [`int`]
- `orientation`: Orientation du navire. [`int`]
- `taille`: Nombre de cases occupées par le navire. [`int`]
- `max_iter`: Le nombre maximal d'itération permis **fixé à 20 par défaut**. [`int`]

Valeur de retour : Orientation aléatoire. [`int`]

Requis :

1. Générer des orientations aléatoires avec `random.randint` et les constantes `NORD`, `SUR`, `EST`, `OUEST` **tant et aussi longtemps que** le navire n'est pas validé par `grille_navire_est_valide()` et que le nombre d'itérations ne dépasse pas la limite égale à `max_iter`.

2. En cas d'échec, déclencher une erreur d'**assertion** avec le message suivant :

```
Nombre d'itérations maximal atteint.
```

xvii. SP17 `grille_ajouter_navire`

Description :

Ajoute un navire à une grille en y insérant les codes appropriés pour la tête et le corps de ce dernier. Le placement d'un navire est déduit à partir de la position de sa tête, de sa taille et de son orientation.

Paramètres :

- `grille`: Grille de jeu. [`list`]
- `navire`: Navire à ajouter. [`dict`]

Valeur de retour : Aucune.

Requis :

1. Extraire l'orientation du navire avec `navire_obtenir_orientation()`.
2. Extraire les positions du navire avec `navire_obtenir_positions()`.
3. La tête du navire – qui est le premier couple `(i,j)` dans la liste de positions – est remplacée par une des constantes `POINTE_NORD`, `POINTE_SUD`, `POINTE_EST` ou `POINTE_OUEST`.
4. Les cellules qui correspondent au corps du bateau sont remplacées par la constante `BATEAU_VIVANT`.

xviii. SP18 `grille_ajouter_flotte_positions_aleatoires`

Description :

Ajoute une flotte vierge à une grille en assignant des positions aléatoires aux navires en s'assurant qu'il n'y a pas de débordements de la grille ni chevauchements de navires.

Paramètres :

- `grille`: Grille de jeu. [`list`]
- `flotte`: Flotte de navires. [`list`]

Valeur de retour : Aucune.

Requis :

5. Générer et assigner une position aléatoire pour la tête de chacun des navires (SP14 et SP15).
6. Générer et assigner une orientation aléatoire qui tient compte de la position de la tête (SP16).
7. Mettre à jour les positions du navire avec `navire_calculer_positions()`.
8. Ajoutez le navire à la grille.

xix. SP19 `grille_gestionnaire_tir`

Description :

Met à jour la grille du joueur par suite d'un tir à une position donnée.

Paramètres :

- `grille`: Grille de jeudu joueur courant. [`list`]
- `i`: Ligne du tir. [`int`]
- `j`: Colonne du tir. [`int`]

Valeur de retour : Aucune.

Requis :

9. Si la cellule visée n'est pas vide (`CELLULE_VIDE`) et n'est pas un tir manqué (`TIR_MANQUE`), la remplacer par `BATEAU_ATTEINT`, sinon par `TIR_MANQUE`.

xx. SP19 `grille_adversaire_gestionnaire_tir`

Description :

Met à jour la grille de l'adversaire par suite d'un tir à une position donnée.

Paramètres :

- `grille`: Grille de jeu de l'adversaire. [`list`]
- `i`: Ligne du tir. [`int`]
- `j`: Colonne du tir. [`int`]
- `est_cible_touchee`: Indique si une cible fut touchée ou non. [`bool`]

Valeur de retour : Aucune.

Requis :

10. Si une cible est touchée, remplacer la cellule par `BATEAU_ATTEINT`, sinon par `TIR_MANQUE`.

12 Procédure principale (`main.py`)

Cette section décrit la procédure principale qui sert à lancer une partie une fois que toutes les autres parties sont terminées. **Faites bien attention à ne pas utiliser de nombres magiques et à faire appel aux fonctions prédéfinies.**

1. Générer une grille vierge pour le joueur.
2. Générer une grille vierge qui représente l'adversaire du point de vue du joueur. Quand le joueur tire, on y notera le résultat du tir pour s'en rappeler plus tard.
3. Générer une grille vierge pour l'ordinateur.
4. Générer une grille vierge qui représente le joueur du point de vue de l'ordinateur. Quand l'ordinateur tire, on y notera le résultat du tir pour s'assurer qu'il ne tire pas deux fois au même endroit plus tard dans la partie.
5. Générer une flotte aléatoire pour le joueur.
6. Générer une flotte aléatoire pour l'ordinateur.
7. Afficher la grille de l'adversaire du point de vue du joueur.
8. Afficher la grille du joueur.
9. Déterminer le joueur courant de manière aléatoire.
10. Démarrez une boucle qui ne s'arrête que lorsque la partie est terminée. **Les étapes 11 à 15 dépendent du joueur courant. Par exemple, si le joueur courant est l'ordinateur, l'adversaire est le joueur.**
    11. Saisir une position sur laquelle tirer.
    12. Déterminer si une cible est touchée avec le gestionnaire de tirs de la flotte.
    13. Afficher l'un des messages suivants en remplaçant `i` et `j` par les bonnes valeurs :

        ```
        Vous avez tiré sur la position (i,j) et avez atteint l'ennemi!
        Vous avez tiré sur la position (i,j) et avez manqué la cible
        L'ennemi à tiré sur la position (i,j) et vous a atteint!
        L'ennemi à tiré sur la position (i,j) et vous a manqué
        ```

    14. Mettre à jour la grille de l'adversaire avec le gestionnaire de tir de la grille.
    15. Mettre à jour la grille de l'adversaire du joueur courant vue par le joueur courant avec le gestionnaire de tir de l'adversaire.
    16. Afficher la grille de l'adversaire du point de vue du joueur.
    17. Afficher la grille du joueur.
    18. Déterminer si la partie est terminée.
    19. Passer au prochain joueur.
20. Afficher le vainqueur.