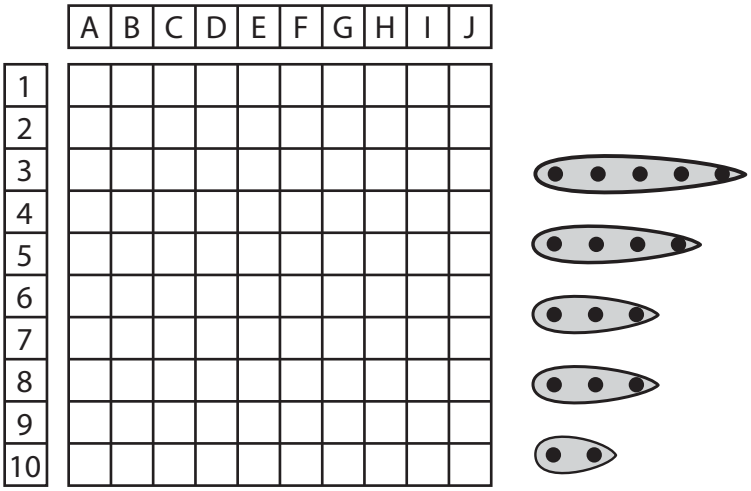
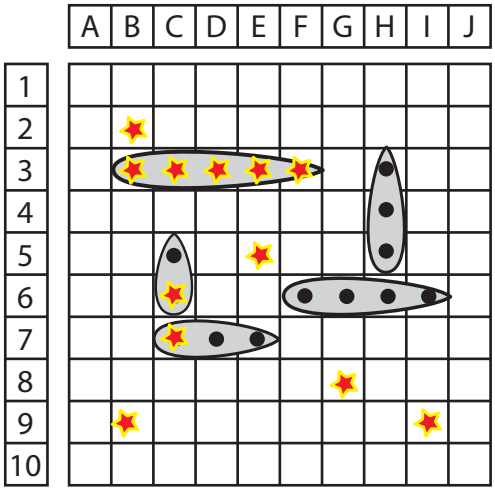
TP1 – IFT2015 – H18 – Major - Les quadtree ou battleship inversé



Dans le jeu battleship (bataille navale) chacun de deux adversaires dispose ses bateaux sur une grille qu'il garde secrète. Tour à tour les joueurs lancent une bombe en spécifiant la coordonnée visée. Le joueur qui reçoit la bombe doit repérer cette coordonnée sur sa grille et déclarer si la bombe touche un bateau (en s'écriant : **touché**) ou non (en s'écriant : **raté**). Chaque bateau occupe un certain nombre de coordonnées contigües (parallèles à l'un des axes) et lorsque chaque coordonnée d'un bateau a été touchée, le bateau est alors déclaré coulé (en s'écriant : **coulé**). La grille à droite montre l'état du jeu d'un des joueurs après quelques bombes larguées par son adversaire. Ici, un vaisseau est coulé et trois vaisseaux au total ont reçu des coups. Le joueur a donc dû dire 5 fois :**raté**, 6 fois : **touché** et 1 fois : **coulé**.



Battleship inversé

Modifions maintenant le jeu :

1) chaque bateau est représenté par un point

2) les bombes larguées détruisent tout bateau dans une zone rectangulaire de taille variable (selon la puissance de la bombe)

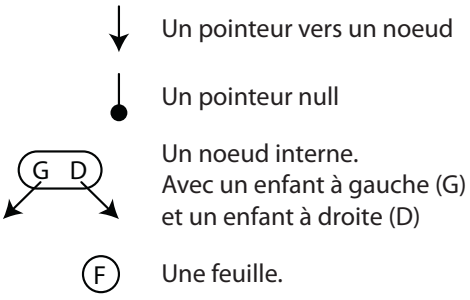
3) le nombre de bateaux n'est pas borné et peut être assez grand (1 million par exemple)

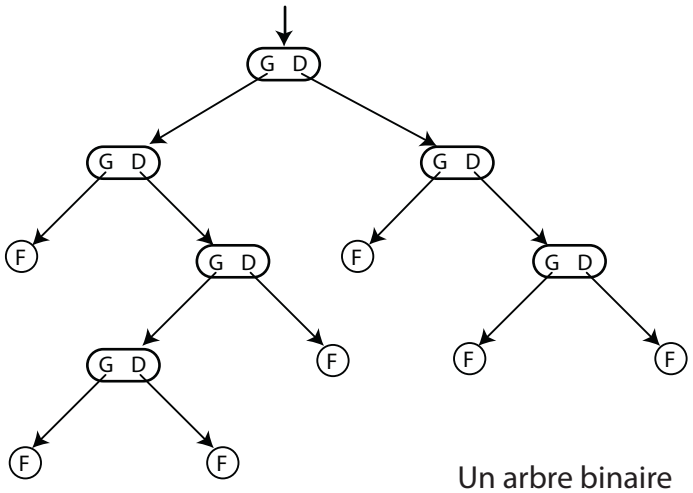
4) la taille de la grille est de 10315 km par 10315 km (soit environ l'équivalent de la surface de l'océan Atlantique)

5) le but du jeu n'est pas de gagner mais de *perdre le plus rapidement et en utilisant le moins de mémoire possible!*

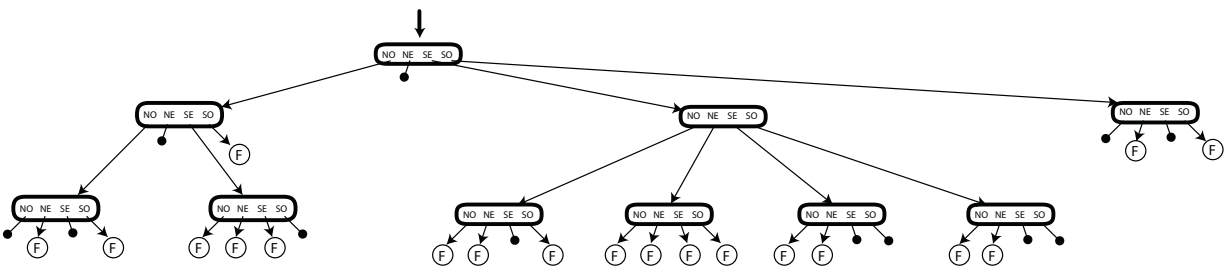
Quadtree - définition

Encodez la position des bateaux dans un *quadtree.* Considérons l'arbre binaire de la figure 4. Chaque nœud est soit une feuille ou un nœud interne avec deux enfants (nommés ici G et D). Un *quadtree* est très similaire mais chacun de ses nœuds compte jusqu'à 4 enfants. Chaque nœud est soit 1) null, 2) une feuille ou 3) un nœud interne. De plus, chaque nœud interne ne peut avoir plus de trois enfants nulls.

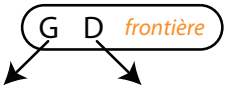




Un arbre binaire

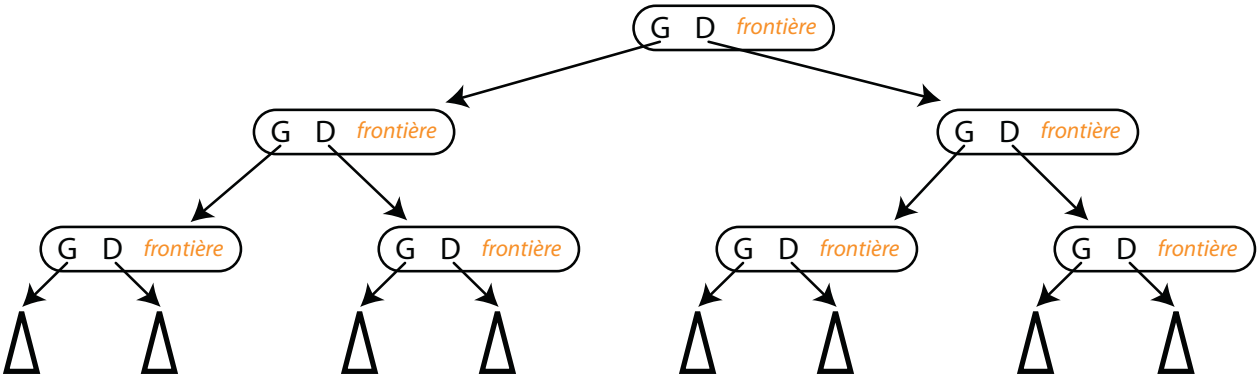


Un quadtree



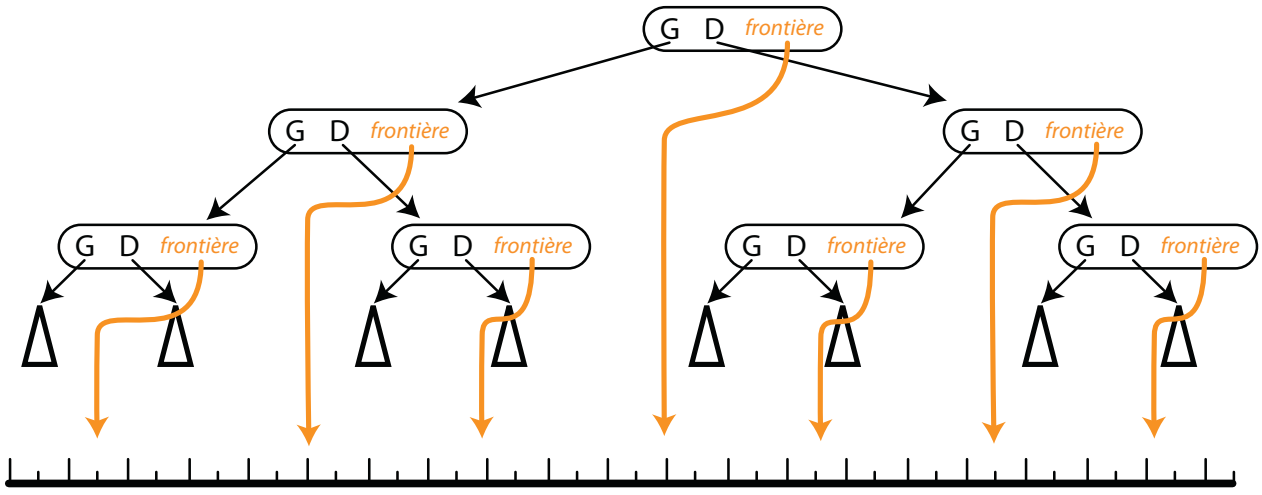
Un noeud binaire interne étiqueté
avec la variable nommée frontière

Un arbre binaire quelconque

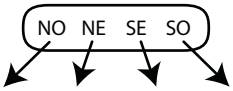


Un arbre binaire étiqueté

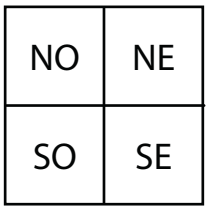
On peut étiqueter un arbre binaire (fig ci-haut) et utiliser cette étiquette pour séparer les points sur une ligne (fig ci-bas).



Un arbre binaire étiqueté avec la valeur frontière gauche/droite sur un trait échelonné



Un noeud quaternaire (quad) avec ses pointeurs sur les quadrants d'un plan

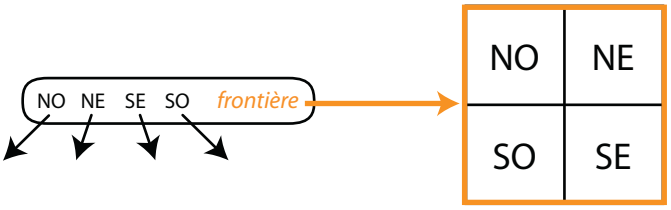


Un plan découpé en quadrants

NO: nord-ouest
NE: nord-est
SE: sud-est
SO: sud-ouest

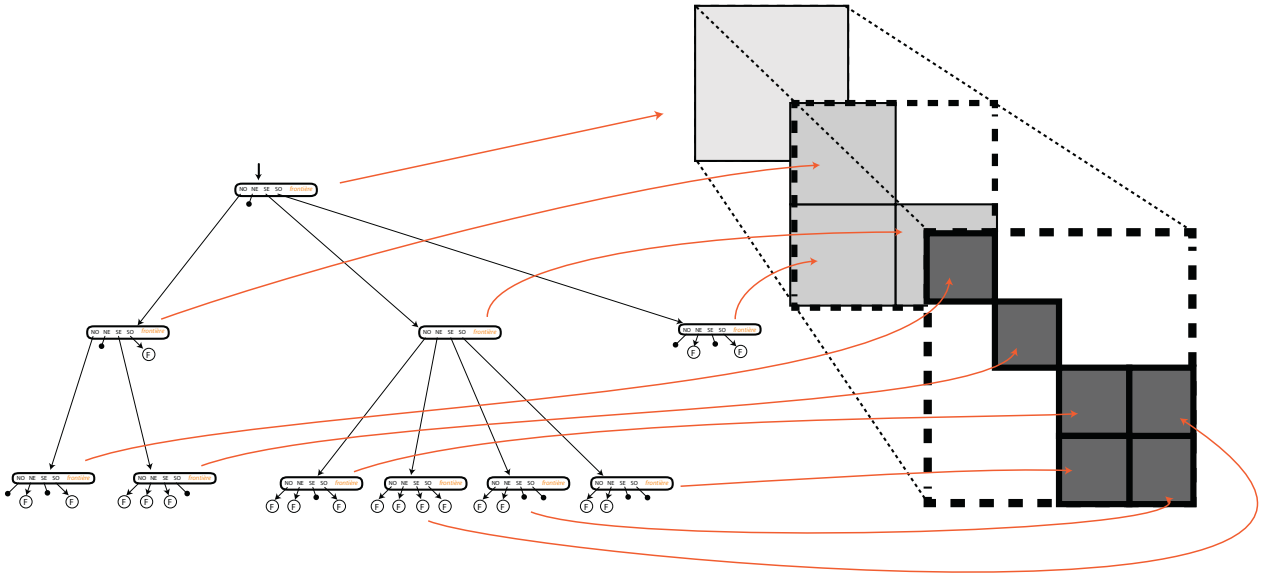
De même, on peut utliser un quadtree pour découper le plan en quatre quadrants : nord-ouest, nord-est, sud-est et sud-ouest.

Le plan qu'un nœud découpe peut-être associé à une variable que l'on nomme ici *frontière*

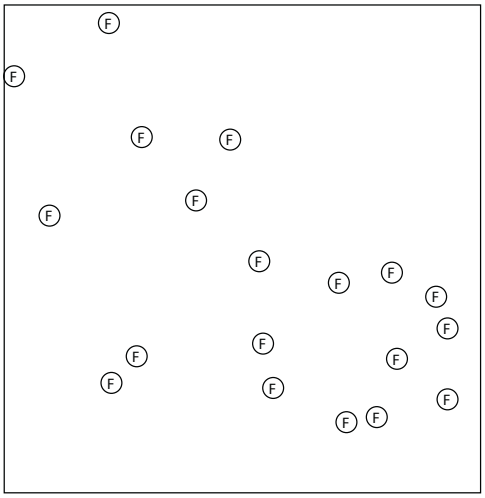


Un noeud quaternaire étiqueté et son plan

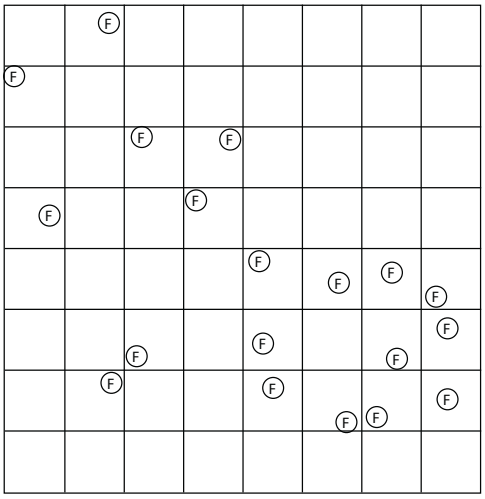
Les subdivisions successives de l'espace par notre quadtree sont montrées ci-dessous. Constatez en particulier qu'il n'est pas nécessaire d'expliciter les sous-arbres pour les quadrants dans lesquels on ne retrouve aucun point. Les quadtrees sont donc naturellement performants pour représenter des ensembles de points clairsemés (« sparse »).



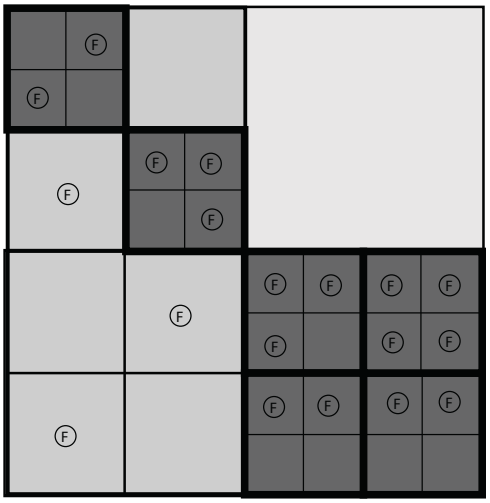
Données sur le plan



Données sur une grille



Données sur le plan partiellement défini du quadtree correspondant



Sur la figure de gauche, l'image du haut montre des points sur un plan cartésien. L'image du centre montre le même espace, sur lequel on a dessiné une grille et l'image du bas montre l'organisation d'un quadtree représentant les mêmes points.

Dans ce TP il vous faut :

Encoder une classe quadtree supportant minimalement les opérations suivantes :

1. Insertion d'une feuille (un bateau)
    a. Une feuille qui est ajoutée peut conduire à l'apparition d'un ou plusieurs nouveau(x) nœud(s) interne(s).
    b. Si une feuille trouve sa place là où un pointeur null est présent, insérer simplement la feuille.
    c. Si une feuille trouve sa place là où une feuille est déjà présente, il faut créer un nouveau nœud interne pour accommoder les deux feuilles. Si toutefois dans ce nouveau nœud interne les deux feuilles occupent le même quadrant (NO par exemple) alors il faudra créer un second nouveau nœud interne. Il peut être nécessaire de répéter ce processus un nombre quelconque de fois.
2. Suppression d'une feuille
    a. Une feuille dont la coordonnée exacte est fournie est retirée de l'arbre. Si le retrait de la feuille conduit à l'apparition d'un nœud interne n'ayant plus que des pointeurs nulls, alors ce nœud devra être retiré de l'arbre.
    b. Puisque le retrait d'un nœud interne peut causer l'apparition d'un autre nœud dont les pointeurs sont tous nulls, cette fonction doit être récursive.
3. Affichage de l'arbre (traversée en largeur)
    a. Chaque niveau de l'arbre est affiché sur une ligne en commençant par le niveau le plus élevé. La première ligne correspond donc à la racine de l'arbre.
    b. Chaque ligne est donnée dans le format suivant :
        i. Un nœud interne est encodé comme suit: <NO NE SE SO>
        ii. NO, NE, SE et SO prennent la valeur 1 s'il existe un enfant associé à ce pointeur ou la valeur 0 si ce pointeur est null.
        iii. Une feuille est encodée comme suit : [X Y]
        iv. X et Y sont des entiers donnant les coordonnées du bateau.
        v. Une ligne est donc une suite de nœuds internes et de feuilles. Par exemple : <0 1 1 0><1 1 1 1>[3298 187]<1 1 0 0>
4. En plus de ces trois méthodes, vous pouvez ajouter d'autres méthodes de votre choix dans le but d'accélérer certaines opérations.
5. Une fonction jouer qui implante les trois phases du jeu décrites ci-après et qui est appelée lorsque le programme est exécuté (fonction main).

Représentation du plan

Le point (0,0) est en haut à gauche du plan (au nord-ouest complètement). L'axe des ***x*** correspond à l'axe ouest-est et l'axe des ***y*** correspond à l'axe nord-sud. Une bombe définit un espace rectangulaire à l'aide de quatre coordonnées de la façon suivante : ***x1*** est la frontière ouest, ***x2*** la frontière est, ***y1*** la frontière nord et ***y2*** la frontière sud. Les frontières sont incluses dans la zone affectée par la bombe.

<u>Les trois phases du jeu :</u>

1. chargement de la position des bateaux dans l'arbre
    a. La liste de coordonnées de bateaux est fournie dans un fichier nommé **bateaux.txt**
    b. Chaque ligne donne les coordonnées d'un bateau dans le format 'x y' où x et y sont des entiers entre 0 et 10315 séparés par une espace.
2. destruction des bateaux par une suite de bombes :
    a. les coordonnées des bombes sont données dans un fichier nommé **bombes.txt**
    b. Chaque ligne donne les coordonnées d'une bombe dans le format 'x1 y1 x2 y2' des entiers (entre 0 et 10315 inclus) séparés par des points tels que x2>=x1 et y2>=y2
    c. Lorsque une bombe est lue, tous les bateaux dont les coordonnées sont dans la portée de la bombe (à l'intérieur de la zone, incluant le périmètre de la zone) sont détruits. C'est à dire qu'ils sont retirés de l'arbre. Après cette opération, il ne doit pas y avoir de nœud dont tous les enfants sont nulls.
3. affichage de l'arbre résiduel
    a. Lorsque toutes les bombes ont été larguées, l'arbre est affiché à l'écran. Celui-ci doit donc contenir tous les nœuds nécessaires, toutes les feuilles nécessaires et aucun nœud dont tous les enfants sont nulls.

Entrée

Votre programme doit comprendre un fichier `tp1_matricule1_matricule2.py` qui sera **exécuté dans un dossier** comprenant les fichiers **bateaux.txt** et **bombes.txt**. Si vous êtes seul vous n'incluez évidemment qu'un matricule. Votre programme ne prend **aucun argument** de ligne de commande.

Sortie
Votre sortie doit être exactement telle que décrite ci-dessus. Votre programme sera évalué sur différentes configurations initiales du jeu.

Tests unitaires
Deux semaines avant la remise, un script pour effectuer des tests unitaires vous sera remis. Par contre, vous devrez écrire vos propres tests unitaires si vous souhaitez vous assurer du fonctionnement correct de votre programme.

Structure du code et version de Python
Avertissement : vous devez développer **VOTRE PROPRE CODE** et tout plagiat détecté entraînera automatiquement un échec.

Votre code doit être clair et bien documenté. Vous êtes encouragés à consulter PEP 8 qui est un guide de style pour Python à la page suivante : https://www.python.org/dev/peps/pep-0008/

Il est fortement recommandé d'utiliser Python 3.6.4 ou toute autre version 3.6.*X*. N'utilisez pas Python 2 car des problèmes de compatibilité pourraient survenir.

Équipes
Vous pouvez faire votre travail seul ou en équipes de **deux** personnes **maximum**. Vous pouvez discuter avec les autres équipes évidemment, mais tout code dupliqué entre deux équipes sera considéré comme un plagiat des deux côtés.

Remise
Vous avez jusqu'au **18 mars à 23h55** pour remettre votre travail sur studium. Remettez votre travail soit sous forme d'un seul fichier `tp1_matricule1_matricule2.py` ou sous la forme d'une archive `tp1_matricule1_matricule2.tar.gz` ou `tp1_matricule1_matricule2.zip` (**aucune** autre forme de compression ne sera tolérée) si vous avez plus d'un fichier. Dans ce cas, l'archive sera décompressée dans le dossier contenant les fichiers de spécifications et devra contenir entre autres le fichier `tp1_matricule1_matricule2.py` qui sera exécuté.

Correction négative

- 50% des points peuvent être perdus si votre code ne s'exécute pas ou ne produit pas de sortie
- Une pénalité de **20% par jour de retard** sera appliquée dès la première seconde de chaque période de 24h, en commençant à 23h56 le 18 mars

Correction positive

- 30% : clarté du code
- 50% : exécution correcte sur un ensemble de tests
- 10% : sortie telle que spécifiée
- 10% : rapidité d'exécution. Si votre vitesse d'exécution correspond à la moyenne, vous aurez tous vos points. Vous en perdrez si votre code est lent, très lent ou extrêmement lent. Si tous les programmes reçus sont jugés efficaces, personne ne perdra de points.

Points boni

Des points boni seront accordés en fonction de la vitesse d'exécution (par rapport à la moyenne) de votre code :

- Code rapide : +4%
- Code très rapide: +7%
- Code extrêmement rapide: +10%

Questions

Envoyez vos questions sur le travail à omailhot92@gmail.com.

Bon travail !!!