Tâche 1 – Bataille navale (battleship) – (100 pts) (A : 20 pts, B : 50 pts, C : 30 pts)

Vous connaissez probablement la bataille navale. Dans ce jeu, chacun des deux joueurs place horizontalement ou verticalement une série de bateaux sur sa zone de combat (grille carrée). Ensuite, chaque joueur à son tour tire une torpille à une coordonnée de la zone de combat de son adversaire qui l'informe si un bateau a été touché. Un bateau n'est coulé que si une torpille a été envoyé sur toutes les cases occupée par celui-ci.

Tâche

Dans cette tâche, on vous demande d'implémenter la phase d'attaque d'une partie de bataille navale dans laquelle votre adversaire n'a qu'un seul bateau de longueur 3 (largeur 1). Vous devez minimiser le nombre de tirs que vous effectuez.

Limites et contraintes

Le bateau se trouve entre les coordonnées 1 et 10 (bornes inclues), horizontalement et verticalement.

	Nombre maximal de tirs
Sous-tâche A	100
Sous-tâche B	50
Sous-tâche C	36

Durée maximale d'exécution : **0.5 seconde**. Limite mémoire : **64 Mo**.

Fonction à implémenter

`play()` : Cette fonction doit faire appel plusieurs fois à la fonction `fire(x, y)` jusqu'à ce que le bateau soit coulé. Une fois coulé, la fonction doit se terminer.

Interface (fonction à appeler)

`fire(x, y)` : Tire à la coordonnée (x, y) et renvoie `true` si le bateau a été touché, `false` sinon.

Exemple

Considérons que le bateau est situé aux coordonnées $(3, 4), (4, 4), (5, 4)$.

Voici une partie possible :

Votre appel	Réponse reçue
$fire(1, 1)$	$false$
$fire(2, 2)$	$false$
$fire(3, 3)$	$false$
$fire(4, 4)$	$true$
$fire(4, 5)$	$false$
$fire(3, 4)$	$true$
$fire(5, 4)$	$true$

Lorsque vous avez touché le bateau à trois coordonnées distinctes, vous pouvez arrêter la partie. Dans cet exemple, le bateau a été coulé en 7 tirs.

Notes pratiques

Cette tâche est interactive. Vous pouvez ne pouvez pas utiliser la compilation et le test automatique de gedit pour cette tâche. Suivez les étapes suivantes pour compiler et tester manuellement votre programme. Demandez de l'aide à un surveillant si nécessaire.

- Ouvrez un terminal (l'application LXTerminal)
- Pour afficher la liste des fichiers dans un répertoire, utilisez la commande `ls`
- Pour changer de répertoire vers le squelette de cette tâche, utilisez la `cd nom_du_répertoire`

Pour ceux qui utilisent C++, les commandes sont les suivantes :

- Compilation : `g++ -std=c++11 -Wall -Werror` *`grader.cpp battleship.cpp`*
- Exécution : `./a.out < input.txt`

Pour ceux qui utilisent Java, les commandes sont les suivantes :

- Compilation : `javac` *`grader.java`*
- Exécution : `java` *`grader`* `< input.txt`

Le résultat de votre programme sera ainsi affiché à la console : "SHIP NOT SUNK" si le bateau n'a pas été coulé et "SHIP SUNK IN X SHOTS" si le bateau a été coulé. Si vous voulez modifier le fichier `input.txt`, sachez qu'il contient les coordonnées des positions du bateau.

Remarques

— Vous ne devez soumettre que le fichier `battleship.cpp` ou `battleship.java`. Il doit implémenter la fonction, `play`, telle que décrite ci-dessus.
— "Résultat incorrect" dans l'interface d'évaluer peut signifier soit que vous avez effectué trop de tirs ou que la fonction `play` s'est terminée sans que le bateau ne soit coulé.
— Si vous continuez à tirer alors que le bateau est coulé et que vous dépassez le nombre de tirs autorisés, votre partie sera considérée comme perdue.
— N'imprimez rien à la console (`print`) dans le programme que vous soumettez.