# Taak 1 – Zeeslag (battleship) – (100 ptn) (A: 20 ptn, B: 50 ptn, C: 30 ptn)

Je kent het spel 'Zeeslag' waarschijnlijk wel. In dat spel moet elke speler horizontaal of verticaal een stel boten plaatsen op zijn/haar slagveld (een vierkant raster). Vervolgens schiet elke speler om beurt een torpedo naar een coördinaat van het slagveld van de tegenstander, die moet zeggen of er raak geschoten werd of niet. Een boot is pas gezonken als alle vakjes die door de boot werden bezet, getorpedeerd zijn.

### Taak

In deze opdracht moet je de aanval van een eenvoudig spel Zeeslag implementeren. Je tegenstander heeft slechts één boot van lengte 3 (en breedte 1). Je moet het aantal schoten dat je afvuurt minimaliseren.

### Limieten en beperkingen

De boot bevindt zich tussen coördinaten 1 en 10 (beiden inbegrepen), horizontaal en verticaal.

| | Maximaal aantal schoten |
|---|---|
| Subtaak A | 100 |
| Subtaak B | 50 |
| Subtaak C | 36 |

Maximale uitvoeringsduur: **0.5 seconden**. Geheugenlimiet: **64 MB**.

### Te implementeren functie:

`play()`: Deze functie moet verschillende keren de functie `fire(x, y)` aanroepen, totdat de boot gezonken is. Eens gezonken, moet de functie eindigen.

### Interface (aan te roepen functie)

`fire(x, y)`: Deze functie vuurt naar coördinaat $(x, y)$ en geeft als resultaat `true` als de boot werd geraakt, en anders `false`.

### Voorbeeld

Stel dat de boot zich bevindt op coördinaten $(3, 4), (4, 4), (5, 4)$.

Dan is dit een mogelijk spelverloop:

| Jouw aanroep | Ontvangen antwoord |
|---|---|
| $fire(1, 1)$ | $false$ |
| $fire(2, 2)$ | $false$ |
| $fire(3, 3)$ | $false$ |
| $fire(4, 4)$ | $true$ |
| $fire(4, 5)$ | $false$ |
| $fire(3, 4)$ | $true$ |
| $fire(5, 4)$ | $true$ |

Zodra je de boot op haar drie verschillende coördinaten hebt geraakt, kan je het spel stoppen. In dit voorbeeld is de boot gezonken na 7 schoten.

### Praktisch

**Dit is een interactieve taak.** Je kan de automatische compilatie-en-test functionaliteit die we aan Gedit hebben toegevoegd niet gebruiken. Volg deze stappen om manueel jouw programma te compileren en te testen. Vraag indien nodig hulp aan een toezichter.

- Open een Terminal-venster (de applicatie LXTerminal).
- Om een lijst bestanden in een map weer te geven, gebruik je het commando `ls`
- Verander van map naar de map met skeletcode voor deze taak met het commando `cd naam_van_map`

Wie C++ gebruikt moet de volgende commando's typen om code te compileren en uit te voeren:

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

Wie Java gebruikt moet de volgende commando's typen om code te compileren en uit te voeren:

- Compileren: `javac grader.java`
- Uitvoeren: `java grader < input.txt`

Het resultaat wordt weergegeven in het terminalvenster: "SHIP NOT SUNKäls de boot niet gezonken is, en "SHIP SUNK IN X SHOTSäls de boot wel gezonken is. Je kan desgewenst het bestand `input.txt` aanpassen, die bevat de gegevens over de posities van de boot.

### Opmerkingen

- Je moet enkel het bestand `battleship.cpp` of `battleship.java` indienen. Die moet de functie `play` implementeren zoals hierboven werd beschreven.
- Als je "Fout resultaat" terugkrijgt van de evaluatie-interface, kan dat zijn omdat je teveel schoten hebt gelost, of omdat de functie `play` werd beëindigd zonder dat de boot is gezonken.
- Als je doorgaat met schieten nadat de boot is gezonken en je gaat voorbij het aantal toegelaten schoten, wordt jouw spel als verloren beschouwd.
- Print niets naar de standaard uitvoer (`print`) in het programma dat je indient.