1 | Uitleg Zeeslag

Bij zeeslag is het de bedoeling dat je alle schepen in een speelveld tot zinken laat brengen. Dat gaat niet zomaar, want de schepen zijn namelijk allemaal verstopt. Je moet dus raden op welke vakjes de schepen staan.

De manier waarop we het spel gaan spelen is via de CLI, wat staat voor Command Line Interface. De CLI kan geen grafische interfaces maken, zoals bijvoorbeeld webpagina's. In plaats daarvan heeft het een veld waar de gebruiker een commando of tekst kan intypen en een output waar regels tekst onder elkaar worden gezet. Voor Zeeslag kunnen we de invoer van de CLI gebruiken om te vragen welke rij en welke kolom de gebruiker wilt raden. Daarna kunnen we het resultaat hiervan in de output zetten (raak of mis).

We zullen het spel zo programmeren dat de gebruiker steeds rijen en kolommen kan opvragen totdat het spel is afgelopen. Dit kan ten eerste door alle schepen te zinken en ten tweede door geen levens meer over te hebben. Levens verlies je door mis te schieten.

2 | Opdracht 1: Game loop

Om ervoor te zorgen dat de gebruiker meerdere keren kan raden, moet de logica voor het spel in een lus worden gezet. Die lus moet uitgevoerd worden totdat ofwel er geen schepen meer zijn ofwel alle levens op zijn.

Opdracht 1a: Zorg dat de while-lus stopt als het spel afgelopen is.

In de while-lus is het eerste wat het programma doet de input van de gebruiker vragen met de klasse Scanner. Maak je hierover niet druk, alles wat je hoeft te weten is dat het cijfer voor de rij opgeslagen wordt in de variabele `row` en de kolom in `column`.

Opdracht 1b: Voordat we het coördinaat kunnen opzoeken, moeten we weten of de rij- en kolomwaarde op het speelveld ligt. Zorg dat het if-statement checkt of zowel `row` als `column` een waarde heeft kleiner dan 0 en groter of gelijk aan 5. Als dat zo is kun je een tekstuitvoer doen naar de CLI dat de waarde foutief is.

Het else-statement zal uitvoeren als de gebruiker een correcte zet heeft gedaan. De tweedimensionale array `guessedLocations` weet voor elk vakje of het wel of niet geraden is, waarbij een 0 niet geraden is en een 1 wel geraden.

Opdracht 1c: Zorg dat het juiste vakje een 1 krijgt in de variabele `guessedLocations`.

In het volgende if-statement besluiten we wat we gaan doen als een schip geraakt is. De variabele `board` heeft net als `guessedLocations` bepaalde informatie voor alle vakjes, alleen `board` geeft per vakje een 0 als er geen schip op staat en 1 als dat wel is.

Opdracht 1d: Zorg ervoor dat als een vakje 1 is, dat je het aantal schepen verlaagt en een melding stuurt naar de gebruiker. Bedenk ook wat er in het else-statement moet komen en maak het af.

Opdracht 1e: Als je het spel nu zou uitvoeren, zou het spel technisch gezien werken. Echter, als het spel is afgelopen weet krijgt de gebruiker niet te weten of hij het spel gewonnen of verloren heeft. Los dit op door na de while-lus een if-statement te plaatsen die dit checkt.

---

3 | Opdracht 2: Het bord laten zien met `displayBoard()`

Het is natuurlijk wel fijn als de gebruiker een visuele weergave heeft van een bord. Zo hoeft hij niet de hele tijd te onthouden welke zetten al gedaan zijn. Een voorbeeld van zo'n visuele weergave is hieronder te zien:

```
  1 2 3 4 5
  ---------
1 |# # # # #
2 |  #  # #
3 |# # # X #
4 |  #  # X
5 |# # # # #
```

Hierbij betekent een X dat een schip is geraakt, een # dat het vakje niet is geraden en leeg als het vakje mis was. Je mag natuurlijk ook proberen om wat mooiers te maken dan in het voorbeeld.

Om alle vakjes van het speelveld te kunnen aflezen, gebruiken we twee in elkaar geneste while-lussen. De eerste is voor de `row` (kolom) en de tweede voor de `column` (rij). Omdat het eerste element van een array altijd het nulde element is, moet je goed rekening houden met de maximale waarde die `row` en `column` in de lus kunnen krijgen. Je wilt dus nooit een vakje opvragen met een `row` van 5, omdat die waarde niet bestaat (de `row` gaat van 0 tot en met 4 en heeft daarom 5 waardes in totaal)!

Let op dat het gaat om twee while-lussen die in elkaar genest zijn. De lussen houden een counter bij van ofwel de rij of de kolom. Merk op dat net voor het einde van elke while lus, de betreffende counter variabele verhoogd wordt met 1.

Opdracht 2a: Verander de maximale waarde/eindwaarde van `row` en `column` in **allebei** de lussen, zodanig dat er 5 `rows` en 5 `columns` zijn waarvoor de lussen zichzelf herhalen.

Nu gaan we kijken welk vakje welk symbool nodig heeft. Om dat te doen moeten we eerst kijken of het vakje geraden is. Zo niet, dan kunnen we een # plaatsen. Anders moeten we kijken of het vakje geraakt is of niet, en dan het juiste symbool neerzetten.

Opdracht 2b: Maak het if-statement compleet. Zorg ervoor dat je het symbool met de juiste methode neerzet met `System.out.print()`. Tip: kijk terug naar je uitwerking van 1c en 1d als je de uitwerking van deze opdracht niet helemaal snapt.

Opdracht 2c: De `displayBoard()` methode werkt nu volledig, hij hoeft alleen nog maar aangeroepen te worden. Doe dit aanroepen op de juiste plek in de while-lus uit opdracht 1.
