1.0.0

H9: Meerdimensionaal werken

N-dimensionale arrays

Organisatievormen van data

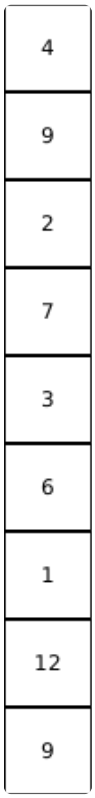
Voorlopig hebben we enkel met 1-dimensionale arrays gewerkt. Je kan er echter ook meerdimensionale maken. In plaats van "reeksen" van data kan je deze zien als tabellen of als figuren.

Als we een array tekenen, doen we dat vaak als een lijn met vakjes op. En als we door een volledige array lopen, doen we dat door de index steeds te verhogen. Om deze reden zeggen we dat arrays een "lineaire" gegevensstructuur zijn, dus "lijnvormig". Een andere manier om dit uit te drukken is "ééndimensionaal".

Een array met de getallen 4, 9, 2, 7, 3, 6, 1, 12 en 9 zouden we ons kunnen inbeelden als volgt:

eendimensionale array, getekend van links naar rechts

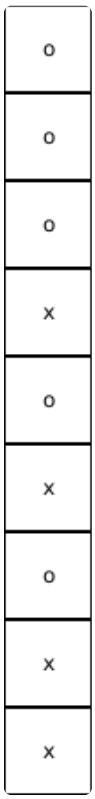
Het maakt niet echt uit of je dit ziet van links naar rechts, van boven naar onder,... Wat wel belangrijk is, is dat je dit ziet als een reeks waar je in een richting door loopt. Onderstaande voorstelling kan dus ook:



eendimensionale array, getekend van boven naar onder

Vaak is het handig om niet met een ééndimensionale voorstelling te werken, maar met een tweedimensionale of driedimensionale voorstelling. Dit kan bijvoorbeeld zijn omdat we data hebben zoals een Excel sheet, een rooster voor het spelletje OXO of vier-op-een-rij, een Sudokupuzzel, een afbeelding (die bestaat uit pixels),...

Een voorbeeld. Kan jij van onderstaande afbeelding in één oogopslag zeggen of er OXO op staat, als de eerste waarde staat voor het vakje linksboven en de laatste voor het vakje rechtsonder?



eendimensionale voorstelling OXO

Niet meteen. Je moet alles eerst "vertalen" in je hoofd naar dit:

Nu kan je veel makkelijker aflezen dat er in de eerste kolom inderdaad OXO staat. Dus de organisatie van je data kan het makkelijker of moeilijker maken bepaalde taken uit te voeren, omdat we visueler kunnen werken.

o	o	o
x	o	x
o	x	x

tweedimensionale OXO

Meerdimensionale arrays leveren hetzelfde voordeel op wanneer je programmeert. Ze staan toe data op te delen in roosters, zodat het makkelijker wordt cellen in een bepaalde richting te doorlopen (rij per rij, kolom per kolom, in de diepte,...). Dit is vaak nuttig om zaken die we "op het zicht" doen te kunnen toepassen op arrays.

> ⓘ Met meerdimensionale arrays kan je technisch niet meer of minder dan met ééndimensionale arrays. Maar afhankelijk van het probleem maken ze het programmeerwerk wel een pak eenvoudiger.

Syntax

De syntax voor een meerdimensionale array is heel gelijkaardig aan die voor een eendimensionale array, maar per dimensie boven de eerste zet je een komma tussen de rechte haken. Bijvoorbeeld:

```
string[,] oxoRooster = new string[3,3];
```

Hierin passen dus 9 strings, maar de extra structuur zal het makkelijker maken rij per rij of kolom per kolom te werken.

Je kan de array ook als volgt initialiseren.

```
string[,] oxoRooster = {{null,null,null},
                        {null,null,null},
                        {null,null,null}}
```

Of hij mag al ingevuld zijn:

```
string[,] oxoRooster = {{"o","o","o"},
                        {"x","o","x"},
                        {"o","x","x"}}
```

Technisch gesproken bevatten de binnenste accolades groepjes waarden in de tweede dimensie van de array en de buitenste ordenen deze groepjes dan in de eerste dimensie. **Maar het volstaat als je onthoudt dat je hier zaken kan noteren alsof ze in rijen en kolommen stonden.**

Je kan nu elementen opvragen door **per dimensie** een index mee te geven. Indexen blijven nog steeds vanaf 0 tellen. Bijvoorbeeld:

```
// toont het vakje in het midden (tweede rij, tweede kolom)
Console.WriteLine(oxoRooster[1,1]);
// toont het vakje rechtsonder (derde rij, derde kolom)
Console.WriteLine(oxoRooster[2,2]);
// toont het vakje onderaan in het midden (derde rij, tweede kolom)
Console.WriteLine(oxoRooster[2,1]);
```

Lengte van iedere dimensie in een n-dimensionale matrix

Indien je de lengte opvraagt van een meer-dimensionale array dan krijg je het totaal aantal posities in de array. Onze OXO-array zal bijvoorbeeld dus lengte 9 hebben. Je kan echter de lengte van iedere aparte dimensie te weten komen met de `GetLength()` methode die iedere array heeft. Als parameter geef je de dimensie mee waarvan je de lengte wenst.

```
int arrayRijen = oxoRooster.GetLength(0);
int arrayKolommen = oxoRooster.GetLength(1);
```

Het aantal dimensies van een array wordt trouwens weergegeven door de `Rank` eigenschap die ook iedere array heeft. Bijvoorbeeld:

```
int arrayDimensions = oxoRooster.Rank;
```

Geneste iteratie

Itereren over een tweedimensionale array

Met meerdimensionale arrays kunnen we de richtingen los van elkaar bekijken. Bijvoorbeeld, om ons OXO-rooster af te printen (eerste rij op één regel, tweede rij op een volgende regel, derde rij op een derde regel), kunnen we dit doen:

```
public static void PrintRij(int rij, string[,] array) {
        for(int i = 0; i < array.getLength(1); i++) {
                Console.Write(array[rij,i]);
        }
        Console.WriteLine();
}

// ergens anders in de code
for(int rij = 0; rij < oxoRooster.GetLength(0); rij++) {
    PrintRij(rij,oxoRooster);
}
```

Als we het rooster in één dimensie zouden voorstellen met een variabele `oxoArray`, moesten we dit doen:

```
// na elementen met index 2, 5 en 8 moeten we een newline printen
for(int i = 0; i < oxoArray.Length; i++) {
    Console.Write(oxoArray[i]);
    // dit omvat meer rekenwerk
    // 3 is hier ook "hardgecodeerd" als lengte van een rij
    // dus als we de spelregels wijzigen (bv. OXOXO), werkt deze code niet meer
    if((i + 1) % 3 == 0) {
        Console.WriteLine();
    }
}
```

We hebben de tweedimensionale versie voor een beter inzicht met een hulpmethode gedaan. Dat hoeft niet, want je kan de inhoud van `PrintRij` ook rechtstreeks invullen. Dan krijg je een **geneste lus**:

```
// per rij doorlopen we de kolommen en printen we een newline
for(int rij = 0; rij < oxoRooster.GetLength(0); rij++) {
    // voor een kolom tonen we gewoon alle waarden
    for(int kolom = 0; kolom < oxoRooster.GetLength(1); kolom++) {
        Console.Write(oxoRooster[rij,kolom]);
    }
    Console.WriteLine();
}
```

In het begin kan deze structuur wat intimiderend zijn, maar er is niets nieuws aan deze luscode. Kijk eerst naar wat één uitvoering doet en zie dat als één geheel. Denk dan na over wat er gebeurt als je dat geheel herhaalt, zonder na te denken over hoe dat geheel werkt.

Itereren met onderling afhankelijke indexen

In het vorige voorbeeld toonden we per rij even veel symbolen. Dus de werking van `PrintRij` hing niet echt af van de kolom waarop we ons bevonden. Dat is niet altijd zo. Soms is er interactie tussen de tellers van de buitenste en de binnenste for-lus.

Volgend voorbeeld toont hoe je enkel een driehoekig deel van een rooster toont:

```
public static void PrintRij(int rij, string[,] array, int maxI) {
        for(int i = 0; i < array.getLength(1) && i <= maxI; i++) {
                Console.Write(array[rij,i]);
        }
        Console.WriteLine();
}

for(int rij = 0; rij < oxoRooster.GetLength(0); rij++) {
    PrintRij(rij,oxoRooster,rij); // print bv. 3 karakters van rij 3, 4 karakters van rij 4,
}
```

Opnieuw kan je de lussen in elkaar schuiven:

```
for(int rij = 0; rij < oxoRooster.GetLength(0); rij++) {
    for(int kolom = 0; kolom < oxoRooster.GetLength(1) && kolom <= rij; kolom++) {
        Console.Write(oxoRooster[rij,kolom]);
    }
    Console.WriteLine();
}
```

En opnieuw hanteer je dezelfde denkwijze om hier uit wijs te raken. Kijk eerst naar wat **in** de lus staat. Zie `rij` gewoon als een getal dat bepaalt hoe veel karakters je maximum kan printen, zonder na te denken over hoe dat getal verandert. Pas als je de werking van de inhoud van de lus goed in je hoofd hebt, denk je na over wat er gebeurt als je die inhoud gaat herhalen met verschillende waarden voor `rij`.

Itereren zonder arrays

Geneste lussen komen vaak voor bij meerdimensionale arrays, maar dat is vooral omdat de twee technieken goed samen gaan. Er is geen verplichting om ze samen te gebruiken.

Je kan geneste lussen gebruiken voor allerlei taken waarbij je herhaling hebt op meerdere niveaus. Bijvoorbeeld voor volgende figuur:

```
1
2 2
3 3 3
4 4 4 4
5 5 5 5 5
```

Hier zijn twee vormen van herhaling:

- de herhaling die het snelst terugkeert, namelijk van links naar rechts (we printen een aantal keer hetzelfde getal)
- de herhaling die minder vaak gebeurt, namelijk van boven naar onder (we printen een aantal keer een regel)

We denken eerst na over de herhaling die het snelst terugkeert. We kennen alles om een getal een aantal keer (gelijk aan zichzelf) te printen:

```
for (int i = 1; i <= getal; i++) {
  Console.Write($"{getal} ");
}
```

We kennen ook alles om een newline een aantal keer te printen:

```
for (int i = 1; i <= anderGetal; i++) {
  Console.WriteLine();
}
```

Om de twee lussen te combineren, moeten we vermijden dat we dezelfde naam `i` gebruiken, dus we hernoemen die van de binnenste herhaling naar `j`. En het aantal keer dat we een getal op een regel willen herhalen, is gelijk aan het nummer van die regel. Dan krijgen we dus:

```
for (int i = 1; i <= anderGetal; i++) {
  for (int j = 1; j <= i; j++) {
    Console.Write($"{i} ");
  }
  Console.WriteLine();
}
```

Oefeningen

Inleiding

Al deze oefeningen maak je als statische methoden van een klasse Meerdimensionaal. Je kan elk van deze methoden uitvoeren via een keuzemenu, zoals bij vorige hoofdstukken.

H09-Som-n-de-rij

Leerdoelen

- Werken met multidimensionale arrays

Functionele analyse

We wensen de gegevens in één rij te groeperen door hun som te bepalen, een beetje zoals je dat in een Excel spreadsheet zou kunnen doen.

Technische analyse

Schrijf in de klasse `Meerdimensionaal` een methode `SomNdeRij`. Deze heeft twee parameters: een meerdimensionale array van `double` en een rijnummer (een `int`). De returnwaarde is de som van alle waarden op de rij met het gevraagde nummer.

Deze methode toont niets op de console, maar je kan ze wel testen via deze voorbeeldcode, die je mag gebruiken wanneer deze oefening wordt opgestart via het keuzemenu:

```
double[,] numbers = {{4.2, 8.1, 3.3},
                     {2.0, 4.0, 6.0},
                     {3.1,3.2,3.3}};
Console.WriteLine("Van welke rij wil je de som zien?");
int row = Convert.ToInt32(Console.ReadLine());
Console.WriteLine(SomNdeRij(numbers,row));
```

H09-Som-per-rij

Leerdoelen

- Werken met multidimensionale arrays

Functionele analyse

We wensen de gegevens in één rij te groeperen door hun som te bepalen, een beetje zoals je dat in een Excel spreadsheet zou kunnen doen.

Technische analyse

Schrijf in de klasse `Meerdimensionaal` een methode `SomPerRij`. Deze maakt een tweedimensionale array met het gevraagde aantal rijen en het gevraagde aantal kolommen aan. Je mag veronderstellen dat er een geldig aantal wordt ingegeven. Vervolgens vraagt ze, rij per rij en kolom per kolom, een getalwaarde. Wanneer elke positie is ingevuld, toont ze de som per rij van de ingegeven getallen.

Voorbeeldinteractie

```
Hoe veel rijen telt je array?
> 3
Hoe veel kolommen telt je array?
> 2
Waarde voor rij 1, kolom 1?
> 4
Waarde voor rij 1, kolom 2?
> 2
Waarde voor rij 2, kolom 1?
> 1
Waarde voor rij 2, kolom 2?
> 1
Waarde voor rij 3, kolom 1?
> 7
Waarde voor rij 3, kolom 2?
> 9
Sommen per rij:
6
2
16
```

H09-Pixels

Leerdoelen

- Werken met multidimensionale arrays
- Werken met enumeratietypes

Functionele analyse

We willen een simpel tekenprogramma maken in de terminal. De gebruiker kan pixel per pixel een gewenste kleur aangeven.

Technische analyse

Schrijf in `Meerdimensionaal` een methode `Pixels`.

Vraag hierin eerst aan de gebruiker welke afmetingen hij wil gebruiken voor zijn afbeelding. Dit bepaalt het aantal rijen en kolommen en dus het aantal pixels. Maak vervolgens een array van `ConsoleColor` waarden aan met deze afmetingen. Vraag tenslotte in een lus wat de gebruiker wil doen:

- een pixel kleuren
  - vraag hierbij de rij-index en kolom-index
  - vraag ten slotte in welke kleur deze moet worden ingevuld
    - je kan sneller een array van alle kleuren krijgen met volgende code: `ConsoleColor[] kleuren = (ConsoleColor[]) Enum.GetValues(typeof(ConsoleColor));`
    - je hoeft deze instructie niet volledig te begrijpen: ze doet hetzelfde als `ConsoleColor[] kleuren = {ConsoleColor.Back, ConsoleColor.DarkBlue, ...}` maar vraagt gewoon minder typwerk
- de afbeelding zoals ze momenteel is tonen
  - toon hiervoor elke pixel als een spatie met `Console.Write(" ")`

Voorbeeldinteractie

H09-color-filter

Leerdoelen

- Werken met multidimensionale arrays
- Werken met enumeratietypes

Functionele analyse

We zullen ons tekenprogramma uitbreiden met een extra functie die de array bewerkt. Met andere woorden, een (simpele) afbeeldingsfilter. Deze zal bepaalde pixels automatisch rood kleuren.

Technische analyse

Schrijf een functie `RedFilter`. Deze heeft een afbeelding (een `ConsoleColor[,]`) als invoer en vervangt alle vakjes die zich op en oneven rij **én** een oneven kolom bevinden door de kleur rood. Ze toont de afbeelding niet.

Deze oefening kan je niet rechtstreeks oproepen via het keuzemenu voor deze klasse. Het wordt een extra optie om iets met je afbeelding te doen.

Voorbeeldinteractie

```
Wat wil je doen?
1. een pixel kleuren
2. afbeelding tonen
3. roodfilter toepassen
```

H09-triangle-filter

Leerdoelen

- Werken met multidimensionale arrays
- Werken onderling afhankelijke indexen

Functionele analyse

We zullen ons tekenprogramma uitbreiden met nog een extra functie die de array bewerkt. Deze functie "knipt" de afbeelding diagonaal in twee.

Technische analyse

Schrijf een functie `LowerTriangleFilter`. Deze heeft een afbeelding (een `ConsoleColor[,]`). Ze zorgt ervoor dat alle pixels op of onder de diagonaal van links boven tot rechts onder wit worden gemaakt. Dit vereist wat te veel rekenwerk als de afbeelding geen vierkant is, dus geef de foutmelding `"deze filter kan niet worden toegepast op deze afbeelding"` als het aantal rijen verschilt van het aantal kolommen.

Deze oefening kan je niet rechtstreeks oproepen via het keuzemenu voor deze klasse. Het wordt een extra optie om iets met je afbeelding te doen.

Voorbeeldinteractie

```
Wat wil je doen?
1. een pixel kleuren
2. afbeelding tonen
3. roodfilter toepassen
4. driehoeksfilter toepassen
```

H09-HeatmapPaardensprong

Leerdoelen

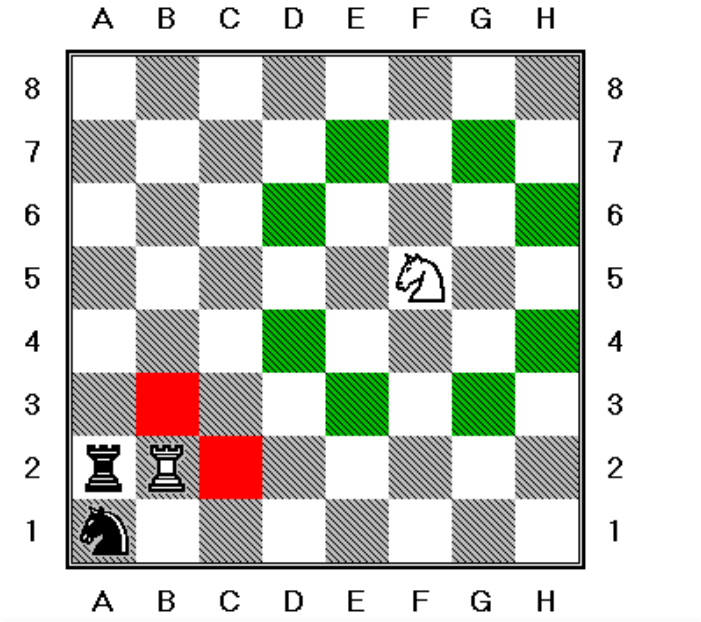
- Werken met multidimensionale arrays
- Werken met geneste iteratie
- Werken met Random

Functionele analyse

Ik wil een paard een aantal willekeurige (random) sprongen laten maken op een schaakbord vertrekkende van een gegeven positie.

Na de sprongen wil ik een "heatmap" van deze sprongen. Hoe vaak is het paard op een positie geweest.

Volg de conventie:



Technische analyse

Schrijf een nieuwe klasse `HeatmapPaardensprong`. Het startpunt in deze klasse is de methode `HeatmapPaardensprongMain`.

Vraag hierin eerst aan de gebruiker hoe vaak het paard moet springen en vraag ook de beginpositie in de vorm A1..H8

Je gaat vervolgens vertrekkende van deze positie een aantal willekeurige sprongen kiezen. Per sprong zijn er maximaal 8 mogelijkheden (zie afbeelding). Je zorgt er voor dat de sprong geldig is (binnen het bord). Deze positie is de startpositie voor de volgende sprong. De positie zelf is ook bezocht, dat houd je bij.

Voorbeeldinteractie

```
82
Hoeveel keer wil je je paard laten springen?
1000
Wat is de initiele positie van het paard (A1..H8)
C3
De heatmap ziet er als volgt uit:
002 009 011 015 011 007 006 011
014 014 010 017 016 014 005 007
007 013 017 014 017 022 020 006
009 022 025 014 021 018 014 010
020 021 017 027 024 022 020 009
011 013 026 035 029 032 022 011
015 015 028 018 021 015 017 011
005 008 019 014 026 014 013 004
```

H09-ConwayGameOfLife

Leerdoelen

- Werken met multidimensionale arrays
- Werken met geneste iteratie
- Werken met Random

Functionele analyse

Surf naar de webstek: https://www.compadre.org/osp/EJSS/3577/12.htm. Hier wordt het "Game of Life" uitgelegd. We gaan een vereenvoudigde console versie maken. Dit wordt bijvoorbeeld veel gebruikt om automatisch omgevingen met structuur te genereren in games met procedurele generatie.

Technische analyse

Schrijf een nieuwe klasse `ConwayGameOfLife`. Het startpunt in deze klasse is de methode `ConwayGameOfLifeMain`.

Vraag hierin eerst aan de gebruiker hoe veel cellen hij wil: 10 geeft een veld van 10 op 10 cellen.

Vervolgens vraag je de gebruiker hoeveel generaties hij wil zien.

Je gaat vervolgens het bord willekeurig opvullen. Gebruik een enum type om de status van een cel (levend, dood) bij te houden.

Vervolgens ga je generatie per generatie laten zien. Van de éne generatie op de andere gelden volgende regels:

- Any live cell with fewer than two live neighbors dies, as if by loneliness
- Any live cell with more than three live neighbors dies, as if by overcrowding.
- Any live cell with two or three live neighbors lives, unchanged, to the next generation.
- Any dead cell with exactly three live neighbors comes to life.

Laat elke generatie 2 seconden zien en bereken dan de volgende generatie (kijk even naar de oefening met de chronometer!).

Voorbeeldinteractie