

# C# OO Programmeren

LES 6

JOHAN DONNÉ

# Demo toepassingen

<https://git.ikdoeict.be/johan.donne/csharp-demo-code>

## C# Demo Code

Deze repository bevat een aantal voorbeeldtoepassingen voor programmatie in c#. Elke toepassing bevat naast de Visual Studio Solution ook een document met beschrijving van de werking en/of gebruikte technieken.

# Random

Patroon:

1 x 'Random' instantiëren

En dan:

'random.Next()' of 'random.Next(10)'

==> Zie democode

# Rekursie

# Herhaling

```
void PrintNumbers(int start, int end)
{
    for (int i = start; i < end; i++)
    {
        Console.Write($"{i},");
    }
    Console.WriteLine (end);
}

...
PrintNumbers(3,6);
```

3, 4, 5, 6

# Recursie

```
void PrintNumbers(int start, int end)
{
    if (start == end)
    {
        Console.WriteLine(end);
    }
    else
    {
        Console.Write($"{start},");
        PrintNumbers(start + 1, end);
    }
}

...
PrintNumbers(3,6);
```

3, 4, 5, 6

# Recursie

Recursie: methode die zich zelf oproept

Steeds:

- recursieve oproep (eenvoudiger probleem)
- basis actie (meest eenvoudige probleem)  
==> in dat geval geen recursieve oproep meer

# Recursie: printNumbers

start = 3,  
end = 6

3,

```
void PrintNumbers(int start, int end)
{
    if (start == end)
    { Console.WriteLine(end); }
    else
    {
        Console.Write($"{start},");
        PrintNumbers(start + 1, end);
    }
}
...
PrintNumbers(3,6);
```



## Recursie printNumbers - stacktrace

start = 3, end = 6	start = 3, end = 6
	start = 4, end = 6

3,

4,

```
void PrintNumbers(int start, int end)
{
    if (start == end)
    { Console.WriteLine(end); }
    else
    {
        Console.Write($"{start},");
        PrintNumbers(start + 1, end);
    }
}
...
PrintNumbers(3,6);
```

## Recursie printNumbers - stacktrace

start = 3, end = 6	start = 3, end = 6	start = 3, end = 6
	start = 4, end = 6	start = 4, end = 6
		start = 5, end = 6

3,

4,

5,

```
void PrintNumbers(int start, int end)
{
    if (start == end)
    { Console.WriteLine(end); }
    else
    {
        Console.Write($"{start},");
        PrintNumbers(start + 1, end);
    }
}
...
PrintNumbers(3,6);
```

# Recursie printNumbers - stacktrace

start = 3, end = 6	start = 3, end = 6	start = 3, end = 6	start = 3, end = 6
	start = 4, end = 6	start = 4, end = 6	start = 4, end = 6
		start = 5, end = 6	start = 5, end = 6
			start = 6, end = 6

3,

4,


5,

6

```
void PrintNumbers(int start, int end)
{
    if (start == end)
    { Console.WriteLine(end); }
    else
    {
        Console.Write($"{start},");
        PrintNumbers(start + 1, end);
    }
}
...
PrintNumbers(3,6);
```

# Recursie printNumbers - stacktrace


start = 3, end = 6	start = 3, end = 6	start = 3, end = 6	start = 3, end = 6	start = 3, end = 6
	start = 4, end = 6	start = 4, end = 6	start = 4, end = 6	start = 4, end = 6
		start = 5, end = 6	start = 5, end = 6	start = 5, end = 6
			start = 6, end = 6	



3, 4, 5, 6

```
void PrintNumbers(int start, int end)
{
    if (start == end)
    { Console.WriteLine(end); }
    else
    {
        Console.Write($"{start},");
        PrintNumbers(start + 1, end);
    }
}
...
PrintNumbers(3,6);
```

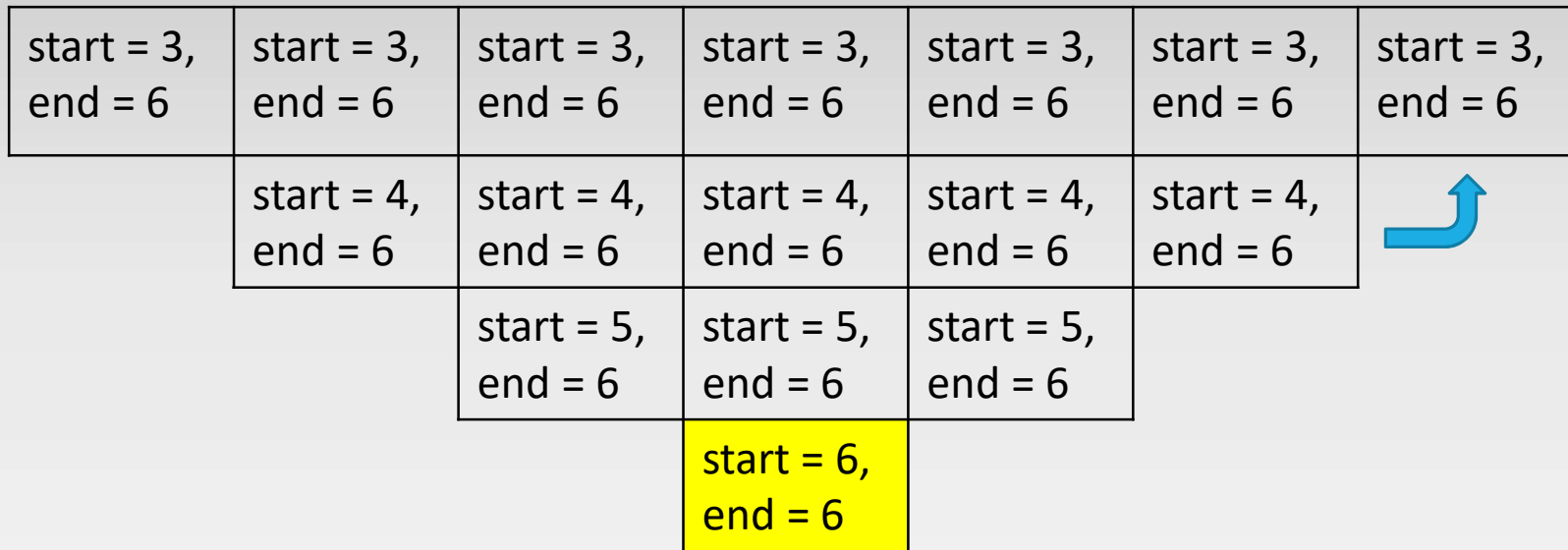
## Recursie printNumbers - stacktrace

start = 3, end = 6	start = 3, end = 6	start = 3, end = 6	start = 3, end = 6	start = 3, end = 6	start = 3, end = 6
	start = 4, end = 6	start = 4, end = 6	start = 4, end = 6	start = 4, end = 6	start = 4, end = 6
		start = 5, end = 6	start = 5, end = 6	start = 5, end = 6	
			start = 6, end = 6		

3, 4, 5, 6

```
void PrintNumbers(int start, int end)
{
    if (start == end)
    { Console.WriteLine(end); }
    else
    {
        Console.Write($"{start},");
        PrintNumbers(start + 1, end);
    }
}
...
PrintNumbers(3,6);
```

# Recursie printNumbers - stacktrace



3, 4, 5, 6

```
void PrintNumbers(int start, int end)
{
    if (start == end)
    { Console.WriteLine(end); }
    else
    {
        Console.Write($"{start},");
        PrintNumbers(start + 1, end);
    }
}
...
PrintNumbers(3,6);
```

# Recurisie – algemene structuur

```
RecursiveMethod(parameters)
{
    if (stopping condition)
    {
        // handle the base case
    }
    else
    {
        // recursive case:
        // possibly do something here
        RecursiveMethod(modified parameters);
        // possibly do something here
    }
}
```

Er zijn meerdere 'base cases' mogelijk

Er zijn meerdere recursieve cases mogelijk

De recursieve oproep brengt ons dichterbij de eindconditie

## Recurisie : machtsverheffing

```
public int Power(int baseNr, int exponent)
{
    if (exponent == 0) return 1;
    else
    {
        return baseNr * Power(baseNr, exponent - 1);
    }
}
...
Console.WriteLine(Power(2,5));
```



# Recursief denken

- Hoe kan het probleem opgesplitst worden in kleinere deelproblemen
- Wat zijn de 'base cases' (en dus eindcondities voor de recursie)
- Hoe worden de deeloplossingen gecombineerd

```
public int Power(int baseNr, int exponent)
{
    if (exponent == 0) return 1;
    else
    {
        return baseNr * Power(baseNr, exponent - 1);
    }
}
```

Recursie kan steeds vervangen worden door herhaling,  
maar is soms (veel) gemakkelijker om te programmeren.

## Recurisie – DirectoryTree

Een console toepassing die een mappenstructuur van je schijf analyseert, en volgende informatie op het scherm toont:

- 'diepte'
- aantal bestanden
- aantal mappen
- aantal gevonden extensies
- de (maximaal) 5 meest voorkomende extensies en het aantal keren dat ze voorkomen in de opgegeven basismap.

Zie <https://git.ikdoeict.be/johan.donne/csharp-demo-code>

# Backtracking

# Backtracking

= 'terugkeren op je stappen'

- Bij zoeken naar een oplossing 'probeert' het programma één keuze uit een aantal alternatieven...
- Probeert of daarmee een oplossing gevonden kan worden...
- Maakt indien nodig de vorige keuze ongedaan om een volgend alternatief te proberen...
- Tot alle alternatieven geprobeerd zijn (of een oplossing gevonden is)
- Meestal in combinatie met recursie

vb. doolhof oplossen

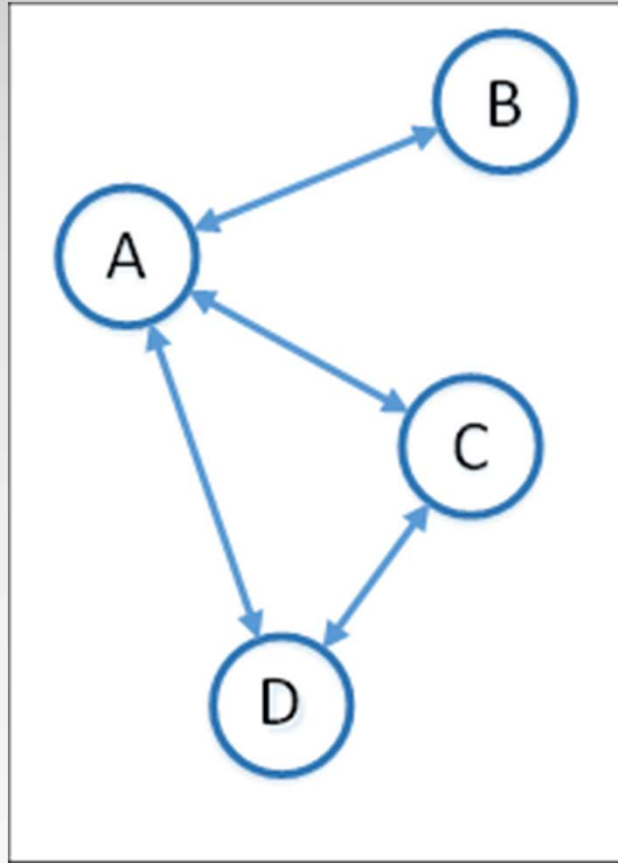
## Backtracking – PathFinding

‘Pathfinding’ is een tak van de artificiële intelligentie (AI) die vaak (maar niet uitsluitend) toegepast wordt bij computergames.

De computer moet dan het optimale pad berekenen tussen twee punten, rekening houdend met eventuele hindernissen, moeilijkheidsgraad van het terrein, enzovoort.

Zie <https://git.ikdoeict.be/johan.donne/csharp-demo-code>

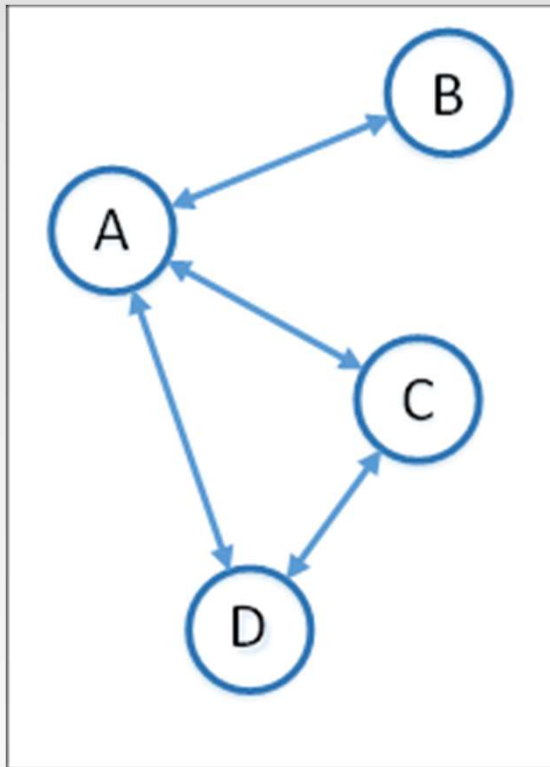
## Backtracking – PathFinding



Hier: op basis van  
'graaf' die informatie  
over locaties en  
verbindingen bevat.

## Backtracking – PathFinding

Interne voorstelling: `Dictionary<string, List<string>>`



Key	Value
A	B, C, D
B	A
C	A, D
D	A, C

# Backtracking – PathFinding

Graaf ingeven via tekstbestand.

Daarna: alle paden laten zoeken tussen twee nodes.

```
PathFinder
```

```
=====
```

```
Graph connections:
```

```
A-B,C,D
```

```
C-D
```

```
-----
```

```
Enter start & destination as <start>-<destination>  
(<enter> to finish):
```

```
> D-B
```

```
Searching path from D to B
```

```
-----
```

```
Number of Paths found: 2
```

```
D - A - B
```

```
D - C - A - B
```

```
>
```



