

Opdracht 1 - Matrix

1	2	3	4	5	6	7	8
9	10	11	12	13	14	15	16
17	18	19	20	21	22	23	24
25	26	27	28	29	30	31	32
33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48
49	50	51	52	53	54	55	56
57	58	59	60	61	62	63	64

- a) Maak een methode:

```
void InitMatrix2D(int[,] matrix)
```

Deze methode vult de binnenkomende matrix met de getallen 1 t/m n .
Hierbij is n het aantal hokjes van de matrix.

- b) Maak ook een methode:

```
void PrintMatrix(int[,] matrix)
```

Deze methode toont het opgegeven array op het scherm.
Gebruik in beide methoden een dubbele for-loop.

→ Test de methoden (aanroep vanuit de Start-methode) met array's met verschillende dimensies (bijvoorbeeld: 8x8, 11x11, 8x10).

- c) Maak een alternatieve init-methode:

```
void InitMatrixLineair(int[,] matrix)
```

Deze methode doet precies hetzelfde als de InitMatrix2D-methode, maar nu gebruik je maar één for-loop met als lopende variabele getal, die de waarde van het te vullen hokje bevat. Deze loopt dus van 1 t/m n (in het voorbeeld hierboven dus van 1 t/m 64). *Je hebt alleen het getal nodig om te bepalen in welke rij/kolom deze geplaatst moet komen.*

→ Test de methoden (aanroep vanuit de Start-methode) met array's met verschillende dimensies.

- d) Maak een alternatieve print-methode:

```
void PrintMatrixWithCross(int[,] matrix)
```

Deze methode print de getallen op de hoofddiagonaal van de matrix **rood** kleurt en geeft de getallen op de andere diagonaal een **gele achtergrond**. Alle andere getallen wit.

Gebruik een geneste lus, en de loop-variabelen (rij en kolom) om de kleur te bepalen (`Console.ForegroundColor` of `Console.BackgroundColor`).

Gebruik `Console.ResetColor` om de kleur te resetten.

1	2	3	4	5	6	7	8	9	10	11
12	13	14	15	16	17	18	19	20	21	22
23	24	25	26	27	28	29	30	31	32	33
34	35	36	37	38	39	40	41	42	43	44
45	46	47	48	49	50	51	52	53	54	55
56	57	58	59	60	61	62	63	64	65	66
67	68	69	70	71	72	73	74	75	76	77
78	79	80	81	82	83	84	85	86	87	88
89	90	91	92	93	94	95	96	97	98	99
100	101	102	103	104	105	106	107	108	109	110
111	112	113	114	115	116	117	118	119	120	121

Opdracht 2 – Zoek getal

- a) Maak een methode:

```
void InitMatrixRandom(int[,] matrix, int min, int max)
```

Deze methode vult het opgegeven array met willekeurige getallen tussen min en max. Gebruik een geneste lus.

```
94 49 50 21 33 40 42 84 30 31
 3 24  1 84 36 62 15 81 84  8
93 69 50 52 81 96 74 45 15 95
94 14 36 47 45 60 16 24  3 69
20 35 27 53 32 98 39 46 47  4
47 80 87 23 98 24 98 98 40 36
66 97 69 35 37 89 88 53 38 75
45 91 78 14 88 57 41 96 19 82
```

- b) Kopieer methode DisplayMatrix van de vorige opdracht.

→ Test de methoden (door deze aan te roepen vanuit de Start-methode).

- c) We gaan nu getallen zoeken in het gegenereerde array. Deze getallen bevinden zich op een positie in het array. Deze positie kunnen we aangeven met een rij en een kolom. Wanneer we echter een methode maken om te zoeken, kunnen we de gevonden positie (twee waarden) niet terug geven als één return waarde. Wanneer we echter een class Positie definiëren, kunnen we wel één Positie teruggeven.

→ Maak een class Positie met twee int-velden: rij en kolom.

- d) Maak een methode:

```
Positie ZoekGetal(int[,] matrix, int zoekGetal)
```

In deze methode zoek je de positie (rij en kolom) in het array matrix. Doorzoek het array van boven naar beneden, van links naar rechts en stop met zoeken bij de eerste positie, waar het gezochte getal zich bevindt. Retourneer een Positie met daarin de rij en kolom van de gevonden positie.

Vraag de gebruiker (in de Start-methode) om een getal in te voeren. Gebruik de methode ZoekGetal om het ingevoerde getal te zoeken en print de gevonden positie.

```
80 48 59 70 88 70 85 15 11 78
 9 95 69 25 22 46 43 94 64 23
98  3 41 70 84 48 95 91 72 76
 7 21  3 82 24  5 81 23 20
51 62 23 33 86 48 42  1 79 73
56 31 23 42  4 76  6 88 85 40
74 93 42 35 70 87 91 86 20 18
41 61 70 44 71 36 80  7 84 35

Geef zoekgetal: 84
Zoekgetal 84 komt het eerst voor op positie [2,4]
Zoekgetal 84 komt het laatst voor op positie [7,8]
```

- e) Implementeer ook een methode:

```
Positie ZoekGetalAchterwaarts(int[,] matrix, int zoekGetal)
```

In deze methode wordt het zoekgetal opgezocht in de array, maar nu van onder naar boven, van rechts naar links; stop het zoeken wederom als het getal gevonden is. Het resultaat zal nu de laatste positie zijn waar het zoekgetal is. Toon ook deze laatste positie op het scherm (in de Start-methode).

Opdracht 3 – Candy Crush

- Maak een `enum RegularCandies`¹ met de volgende opties: JellyBean (red), Lozenge (orange), LemonDrop (yellow), Gum Square (green), LollipopHead (blue), Jujube Cluster (purple).
- Maak in de Start-methode een 2-dimensionale array van het type `RegularCandies` (dus een matrix waarin de regular candies geplaatst kunnen worden). Geef dit array de naam speelveld.



- Maak een methode:
`void InitCandies(...)`
 waarbij het 2-dim array wordt meegegeven als parameter. Vul het array met random candies, waarbij alle regular candies gebruikt kunnen worden. Je kunt hier slim gebruik maken van het omzetten van een int-waarde naar de bijbehorende enum-waarde.

- Maak een methode:
`void PrintCandies(...)`
 waarbij het 2-dim array weer wordt meegegeven als parameter. Deze print-methode gebruikt voor elke candy een andere kleur. De standaard kleur kun je weer herstellen met `Console.ResetColor()`.



- Maak een methode
`bool ScoreRijAanwezig(...)`
 met als parameter het 2-dim array. Deze methode geeft `true` terug als er 3 (of meer) symbolen naast elkaar staan in een rij, anders `false`. Ga dus alle rijen (één voor één) bij langs, onthoudt de 'huidige candy' en zet een teller op 1. Als de volgende candy gelijk is (aan de vorige) verhoog dan de teller, anders reset de teller naar 1 en stel de 'huidige candy' opnieuw in.
 → Controleer of er 3 (of meer) symbolen naast elkaar staan in een rij.
- Maak ook een methode
`bool ScoreKolomAanwezig(...)`
 Deze controleert of er 3 (of meer) symbolen onder elkaar staan.

¹ <http://www.candycrusher.com/candy-crush-symbol-meanings/>

Opdracht 4 – Paardensprong (*niet verplicht*)

- a) Maak een 2-dimensionale int-array 'schaakbord' aan met 8 rijen en 8 kolommen. De waarden in de hokjes van dit array betekenen het volgende:
0 = 'leeg', 1 = 'bezet', 2 = 'mogelijke zet'.




- b) Maak een methode:

```
void InitSchaakbord(int[,] schakbord)
```

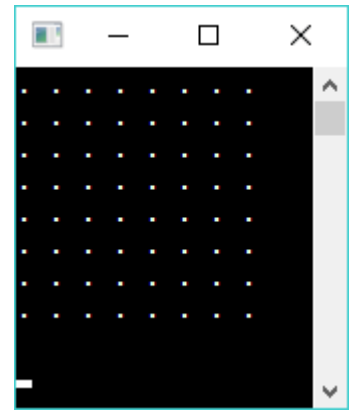
Deze methode maakt alle hokjes van het schaakbord leeg (waarde 0)

- c) Maak een methode:

```
void ToonSchaakbord(int[,] schakbord)
```

Deze methode toont het schaakbord op het scherm. (0 =  1 =  2 = )

→ Test de methoden (aanroep vanuit de Start-methode).

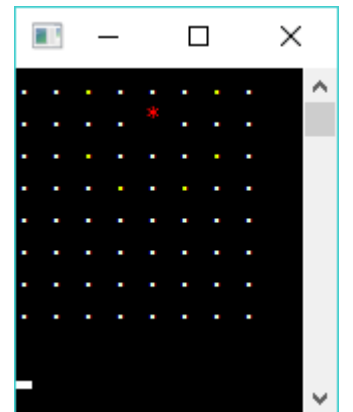


- d) Maak een methode:

```
Positie PlaatsPaard(int[,] schakbord)
```

Deze methode plaatst een Paard (het schaakstuk) op een random positie van het bord (waarde 1) en geeft de positie terug (gebruik een class Positie).

→ Test de methoden (aanroep vanuit de Start-methode).



- e) Maak een methode:

```
void MogelijkePaardenSprongen(int[,] schakbord, Positie positie)
```

Deze methode bepaalt de mogelijke posities waar het Paard naar toe kan springen (waarde 2). Je doet dit natuurlijk op een slimme manier (b.v. met een loop). Let op, er zijn niet altijd 8 mogelijkheden!

De parameter positie geeft aan waar het paard staat.

→ Test de methoden (aanroep vanuit de Start-methode).

