

---

## Pygame of Life

Koen De Bruyne, CoderDojo Ichtegem





# Inhoudstafel

<b>1</b>	<b>Pygame of Life</b>	<b>1</b>
1.1	Info: Conway's Game of Life . . . . .	1
1.2	Het spel: eerst en vooral . . . . .	4
1.2.1	Python . . . . .	4
1.2.2	Editor . . . . .	4
1.2.3	Pygame installeren . . . . .	4
1.3	Speelveld . . . . .	5
1.3.1	Import . . . . .	5
1.3.2	Grootte en snelheid . . . . .	5
1.3.3	pygame . . . . .	5
1.3.4	Spellus . . . . .	5
1.3.5	Raster . . . . .	6
1.4	Het leven begint . . . . .	7
1.4.1	De eerste fase tekenen . . . . .	7
1.4.2	De regels van het leven in code . . . . .	9
1.4.3	De regels toepassen . . . . .	11
1.4.4	Willekeurige start . . . . .	13
1.4.5	Kleinere cellen of snellere evolutie . . . . .	13



# 1 Pygame of Life

De code is ook terug te vinden op <https://github.com/koen-db/PygameOfLife>

## 1.1 Info: Conway's Game of Life

*Game of Life*<sup>1</sup>, soms kortweg *Life* genoemd, is een in 1970 door de Britse wiskundige John Conway bedachte cellulaire automaat, een tweedimensionaal raster met vierkante 'cellen' die 'levend' of 'dood' kunnen zijn, en die zich volgens vastgestelde regels ontwikkelen en daarbij allerlei patronen kunnen vormen.

---

<sup>1</sup> Game of Life, Wikipedia

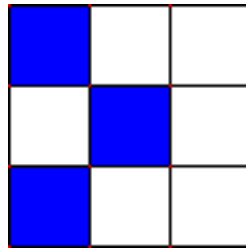


**Figure 1.1:** By Thane Plambeck - "<https://www.flickr.com/photos/thane/20366806/>", CC BY 2.0, <https://commons.wikimedia.org/w/index.php?curid=13076802>

Er wordt gewerkt op een raster van vierkante cellen. Het raster is onbegrensd, dus oneindig groot. Elke cel kan zich in twee toestanden bevinden: een cel is gekleurd of wit.

Nadat een aantal cellen van een raster is ingekleurd, begint het spel. Game of Life werkt met ‘generaties’; om te bepalen of een cel gekleurd (‘levend’) is of wit (‘dood’) in de volgende generatie wordt er een aantal regels toegepast aan de hand van de status (levend of dood) die de buurcellen hebben. Elke cel heeft 8 buurcellen.

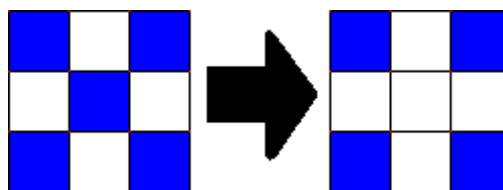
- Als een cel door 2 of 3 gekleurde buurcellen omgeven wordt, blijft deze cel zelf ook gekleurd, zoals in het voorbeeld hieronder.



**Figure 1.2:** Blijvenleven.png

In deze afbeelding blijft de middelste cel gekleurd, want de cel wordt omgeven door 2 andere gekleurde cellen.

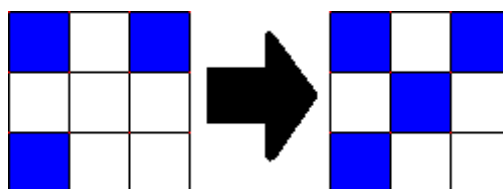
- Als een cel door 4 of meer gekleurde buurcellen omgeven wordt, gaat deze cel dood door 'overbevolking' (dat wil zeggen, de cel wordt wit). Als een cel door minder dan twee gekleurde buurcellen omgeven wordt, gaat deze cel ook dood, maar dan door eenzaamheid. Zie voorbeeld hieronder.



**Figure 1.3:** Doodgaan.png

In deze afbeelding gaat de middelste cel dood, want de cel wordt door meer dan 3 of minder dan 2 gekleurde cellen omgeven.

- Als een dode cel wordt omgeven door precies 3 gekleurde buurcellen, wordt deze dode cel ook gekleurd ('geboren'), zoals in het voorbeeld hieronder.



**Figure 1.4:** Geborenworden.png

In deze afbeelding wordt de middelste cel geboren, want de cel wordt door exact 3 gekleurde cellen omgeven. Al deze transformaties geschieden gelijktijdig voor alle cellen.

## 1.2 Het spel: eerst en vooral

Voor we van start gaan hebben we een aantal zaken nodig. Die zul je op je computer moeten installeren.

### 1.2.1 Python

Python zelf uiteraard: <https://www.python.org/downloads/>

### 1.2.2 Editor

Een programma om code te schrijven, dit kan de meegeleverde REPL zijn of een andere editor.

### 1.2.3 Pygame installeren

Pygame is een bibliotheek aan code om eenvoudig spelletjes mee te maken, zonder veel code zelf te moeten schrijven.

Je kan dit installeren door in de *commandoprompt* het volgende in te tikken:

```
pip install pygame
```

Je bekomt een *commandoprompt* door in Windows het Windows-symbool samen met de R toets in te duwen, en daarna cmd in te tikken.

```
pip install pygame
```

```
Collecting pygame
```

```
Obtaining dependency information for pygame from https://files.pythonhosted.org/p
2.5.2-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata
```

```
Downloading pygame-2.5.2-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_64
```

```
Downloading pygame-2.5.2-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_64.v
```

```
14.0/14.0 MB 18.6 MB/s eta 0:00:00
```

```
Installing collected packages: pygame
```

```
Successfully installed pygame-2.5.2
```

```
[notice] A new release of pip is available: 23.2.1 -> 23.3.1
```

```
[notice] To update, run: pip install --upgrade pip
```



## 1.3 Speelveld

### 1.3.1 Import

Om ons spel te maken gaan we de code inladen die pygame voor ons voorziet.

```
import pygame
```

### 1.3.2 Grootte en snelheid

We maken een aantal variabelen: eentje voor de grootte van het scherm (in pixels) en eentje voor de snelheid: 10 frames per seconde.

```
SCHERM = BREEDTE, HOOGTE = 1280, 720  
SNELHEID = 10
```

### 1.3.3 pygame

We starten daarna pygame op, maken een speelveld met de SCHERM parameter en maken een klok die de tijd van het spel zal bijhouden.

```
pygame.init()  
speelveld = pygame.display.set_mode(SCHERM)  
klok = pygame.time.Clock()
```

### 1.3.4 Spellus

De spellus is een eindeloze lus, die dus de ganse tijd herhaalt. Die maken we door `while True:` op een lijn te zetten.

In Python zetten we de lijnen eronder telkens met 4 spaties ervoor, zolang die lijnen in de lus moeten uitgevoerd worden.

- `speelveld.fill` zal het kader opvullen met zwart.
- `for event...` zorgt ervoor dat we het spel kunnen afsluiten, let hier ook telkens op de 4 spaties telkens er een nieuwe lijn komt.
- `pygame.display.flip` zal het scherm tekenen
- `klok.tick()` zegt dan weer tegen het spel dat er een volgende fase mag ingegaan worden.

```

while True:

    speelveld.fill(pygame.Color('black'))
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            exit()

    pygame.display.flip()
    klok.tick(SNELHEID)

```

### 1.3.5 Raster

Om het spel duidelijker te kunnen zien gaan we een *raster* tekenen. Deze waarde komt vlak na de SNELHEID en voor de `pygame.init()` te staan:

- TEGEL zal de grootte zijn in pixels van elk vierkantje van de raster

```

...
SNELHEID = 10
TEGEL = 40

pygame.init()
...

```

Daarna tekenen we de raster.

! let op de insprong van deze lijnen, die moet even ver naar rechts staan als de `pygame.display.flip()`

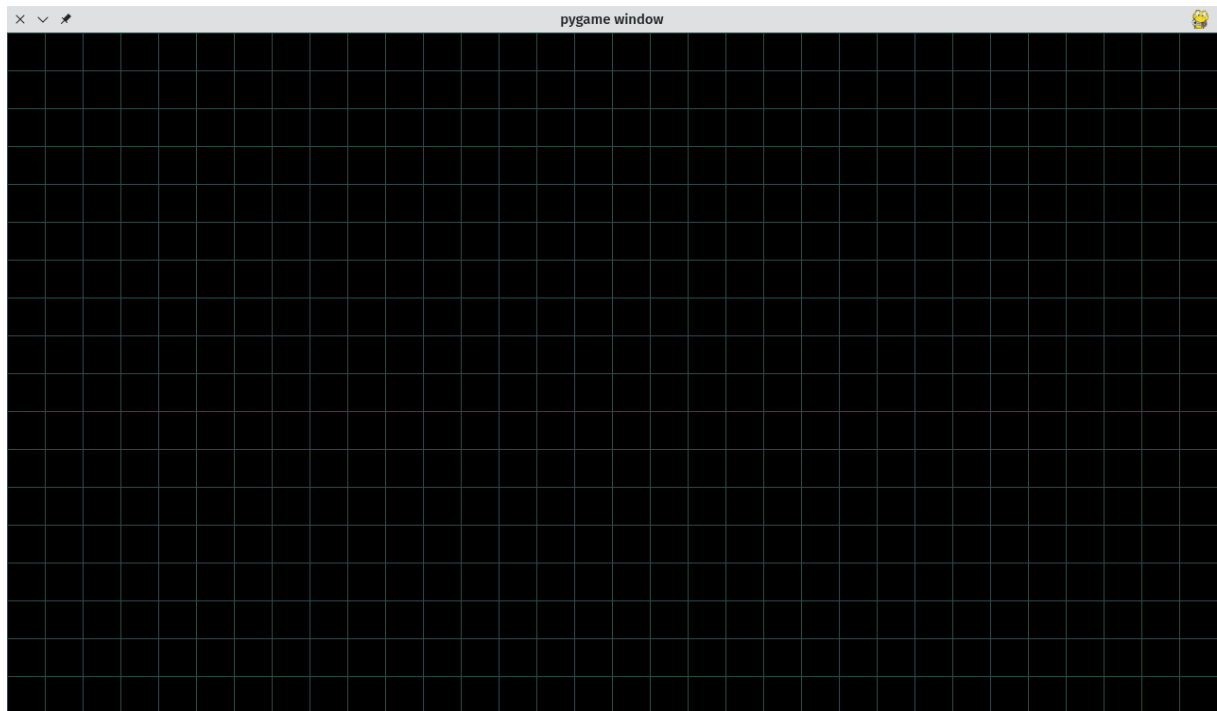
```

...
    exit()

    [pygame.draw.line(speelveld, pygame.Color('darkslategray'), (x, 0), (x,
↪ HOOGTE)) for x in range(0, BREEDTE, TEGEL)]
    [pygame.draw.line(speelveld, pygame.Color('darkslategray'), (0, y),
↪ (BREEDTE, y)) for y in range(0, HOOGTE, TEGEL)]

    pygame.display.flip()
    ...

```



**Figure 1.5:** raster.png

## 1.4 Het leven begint

### 1.4.1 De eerste fase tekenen

Wij gaan *leven* voorstellen als groene vierkantjes in de raster.

We nemen het aantal kolommen (K) en rijen (R) door de BREEDTE en HOOGTE van het speelveld te delen (//) door het aantal pixels van een tegel.

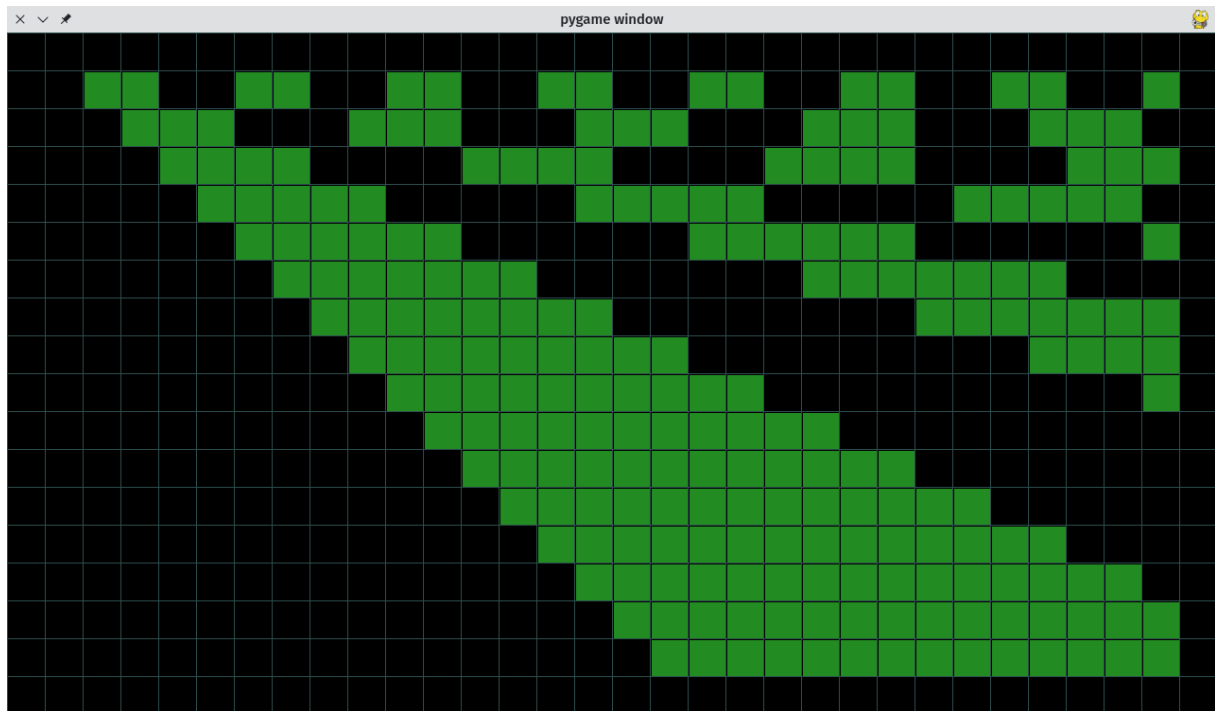
```
K, R = BREEDTE // TEGEL, HOOGTE // TEGEL
```

De volgende code maakt een raster zo groot als ons speelveld en vult dit op met 0 en 1. Let goed op met het overnemen van de formule, een klein foutje is snel gemaakt. Het is een beetje een willekeurige formule die een leuk resultaat geeft.

```
huidig_veld = [ [(k // (r+1)) % 2 for k in range(K)] for r in range(R)]
```

Dit maakt de volgende tekening:



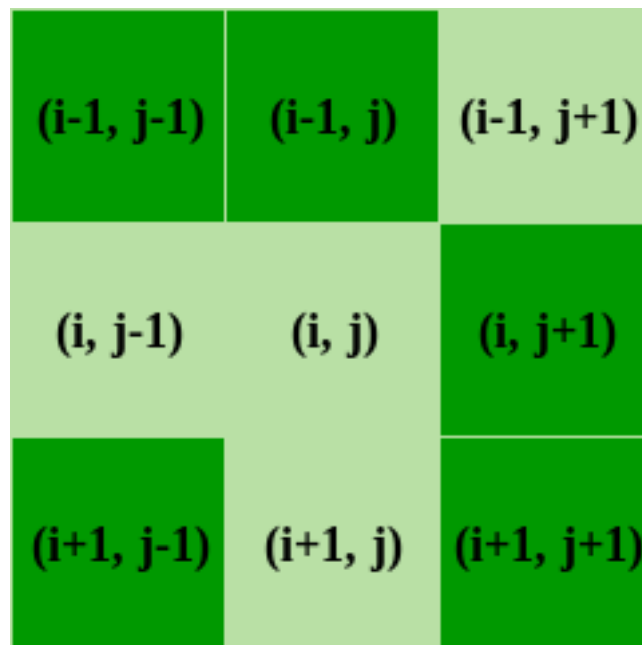


**Figure 1.6:** patroon.png

### 1.4.2 De regels van het leven in code

Een cel in ons raster kan 2 statussen hebben: “AAN” (1) of “AF” (0). De regels kijken altijd naar een cel in verhouding tot de 8 omliggende cellen. Cellen veranderen van status tussen beurten in, volgens deze 4 regeltjes:

1. Als een cel AAN is en minder dan 2 burens heeft die ook AAN zijn, gaat de cel AF
2. Als een cel AAN is en 2 of 3 burens heeft die AAN zijn, blijft de cel AAN
3. Als een cel AAN is meer dan 3 burens heeft die AAN zijn, gaat de cel AF
4. Als een cel AF is en exact 3 burens heeft die AAN zijn, gaat de cel AAN



**Figure 1.7:** regels.png

In python code wordt dit dan als volgt geschreven. Voor elke cel in `huidig_veld` wordt volgens de positie `x` en `y` gekeken naar de status van de omliggende burenen. Als de status AAN ofwel 1 is, wordt de teller `aantal_buren` met 1 verhoogd. Dit geheel is wat een functie genoemd wordt.

... + 2 ziet er vreemd uit, maar dat is omdat `range()` telt “tot” dat getal en niet “tot en met” dat getal.

Eenmaal we het aantal actieve burenen hebben, kijken we naar de status van de cel zelf en hoeveel burenen er actief zijn. Aan de hand daarvan passen we de status van de cel zelf aan.

```
def regels(huidig_veld, x, y):
    aantal_buren = 0
    for j in range(y - 1, y + 2):
        for i in range(x - 1, x + 2):
            if huidig_veld[j][i]:
                aantal_buren += 1

    if huidig_veld[y][x]:
        aantal_buren -= 1
        if aantal_buren == 2 or aantal_buren == 3:
            return 1
        return 0
    else:
        if aantal_buren == 3:
```

```
        return 1
    return 0
```

### 1.4.3 De regels toepassen

We starten met een volledige leeg veld voor de volgende “beurt”:

```
volgend_veld = [[0 for k in range(K)] for r in range(R)]
```

Tijdens het tekenen van het huidige veld, passen we ook de regels toe voor het volgende veld:

```
volgend_veld[y][x] = regels(huidig_veld, x, y)
```

Daarna kopiëren we het volgende veld naar het huidige speelveld. Dit gebeurt nadat alles getekend werd.

```
from copy import deepcopy
# Het 'volgende' veld wordt nu het 'huidige' veld
huidig_veld = deepcopy(volgend_veld)
```

Voor de duidelijkheid het complete programma van voor naar achter:

```
import pygame
from copy import deepcopy

SCHERM = BREEDTE, HOOGTE = 1280, 720
SNELHEID = 10
TEGEL = 40
K, R = BREEDTE // TEGEL, HOOGTE // TEGEL

pygame.init()
speelveld = pygame.display.set_mode(SCHERM)
klok = pygame.time.Clock()

huidig_veld = [(k + r) % 2 for k in range(K-1)] for r in range(R-1)]
volgend_veld = [[0 for k in range(K)] for r in range(R)]

def regels(huidig_veld, x, y):
    aantal_buren = 0
```

```

    for j in range(y - 1, y + 2):
        for i in range(x - 1, x + 2):
            if huidig_veld[j][i]:
                aantal_buren += 1

    if huidig_veld[y][x]:
        aantal_buren -= 1
        if aantal_buren == 2 or aantal_buren == 3:
            return 1
        return 0
    else:
        if aantal_buren == 3:
            return 1
        return 0

while True:

    speelveld.fill(pygame.Color('black'))
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            exit()

    [pygame.draw.line(speelveld, pygame.Color('darkslategray'), (x, 0), (x,
↪ HOOGTE)) for x in range(0, BREEDTE, TEGEL)]
    [pygame.draw.line(speelveld, pygame.Color('darkslategray'), (0, y),
↪ (BREEDTE, y)) for y in range(0, HOOGTE, TEGEL)]

    for x in range(1, K - 1):
        for y in range(1, R - 1):
            if huidig_veld[y][x]:
                pygame.draw.rect(speelveld, pygame.Color('forestgreen'), (x
↪ * TEGEL + 2, y * TEGEL + 2, TEGEL - 2, TEGEL - 2))
                volgend_veld[y][x] = regels(huidig_veld, x, y)

    # Het 'volgende' veld wordt nu het 'huidige' veld
    huidig_veld = deepcopy(volgend_veld)

    pygame.display.flip()
    klok.tick(SNELHEID)

```



#### 1.4.4 Willekeurige start

Bovenaan importeren we de keuzefunctie `choice` die ons zal toelaten een willekeurige 1 of 0 te kiezen:

```
from random import choice
```

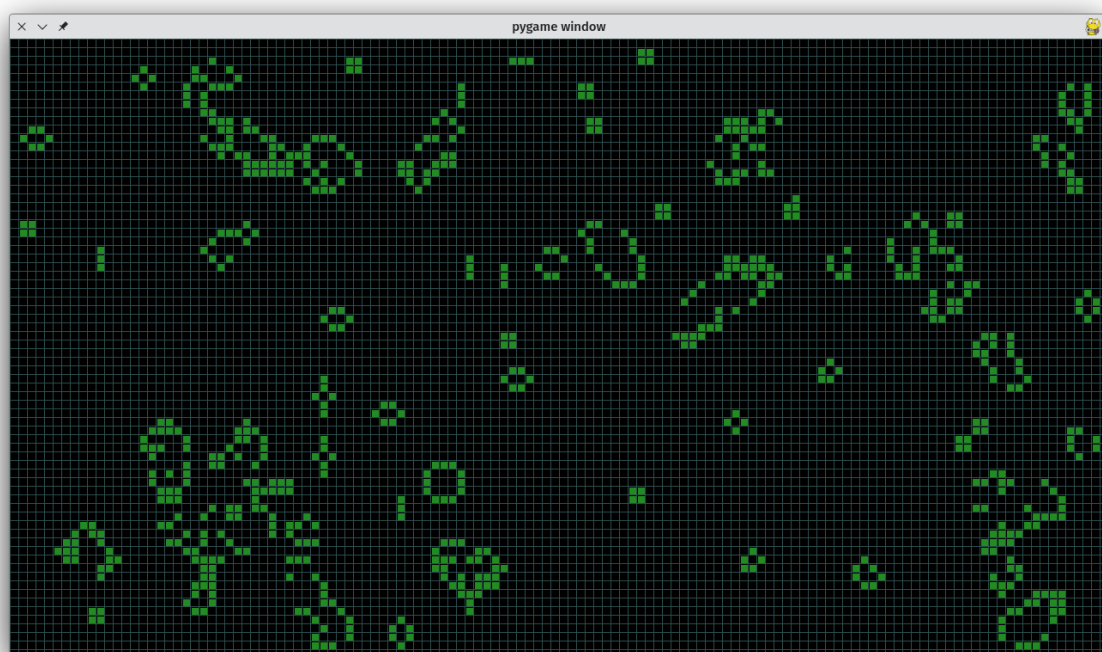
Daarna passen we de lijn aan die het initiële veld maakt:

```
huidig_veld = [[choice([0, 1]) for k in range(K)] for r in range(R)]
```

#### 1.4.5 Kleinere cellen of snellere evolutie

Speel eens met de waarde van `TEGEL`, probeer bijvoorbeeld eens 20 of 10.

Speel ook eens met de waarde van `SNELHEID`, probeer dit eens hoger te zetten om snellere evolutie te krijgen.



**Figure 1.8:** random.png

Bronvermelding:

Dit spelletje is gebaseerd op <https://github.com/StanslavPetrovV/Python-Game-of-life/> (MIT licentie). Dezelfde code werd gebruikt in een Youtube video: [https://www.youtube.com/watch?v=lk1\\_h2\\_GLv8](https://www.youtube.com/watch?v=lk1_h2_GLv8)