Blok 2 - Retro gaming with Python

- Vallende stenen
- Snake
- Introductie AI
- Snake Challenge
- Kennis Check Blok 2
- Snake lesmatriaal

Vallende stenen

Cursus: 28567

0 Wat gaan we leren

datacourse

We gaan nog een projectje maken met Thonny (uit de vorige les) en bij dit project gaan we gebruik maken van de standaard Python library:

pgzero

Weet je nog hoe je een package installeert in Thonny?

Yep, Tools - Manage Packages en dan **pgzero** zoeken en installeren.

image.png

?? Opdracht

Waarvoor dient **pgzero**?

Hint: toen je de package opzocht in Thonny zag je een omschrijving.

? Inleveren

Korte omschrijving in één of twee zinnen waarvoor jij denkt dat je het package **pgzero** heb moeten installeren.

1 Teken speler en steen

In deze les gaan we de speler en één vallende steen tekenen.

We beginnen met een eenvoudige versie: een blokje onderaan dat je straks kunt besturen, en een steen die we straks laten vallen.

Wat gaan we doen?

We tekenen een rechthoek voor de speler en een rechthoek voor de steen.

We gebruiken vaste posities om de eerste versie werkend te krijgen.

Weet je nog hoe de posities in Pygame werken?

(0,0) staat linkbovenaan, de eerste coordinaat laat zien hoe ver je naar rechts gaat en de tweede how ver je naar bedene gaat.

image.png

? Code

```
import pgzrun

WIDTH = 800
HEIGHT = 600

# Speler onderaan het scherm
player_x = 400
player_y = 550
player_width = 80
player_height = 20

# Vallende steen bovenaan
rock_x = 300
rock_y = 0
rock_size = 40

def draw():
    screen.clear()
    screen.draw.filled_rect(Rect((player_x, player_y), (player_width, player_height)), "blue")
    screen.draw.filled_rect(Rect((rock_x, rock_y), (rock_size, rock_size)), "gray")

pgzrun.go()
```

?? Uitleg

- `player_x` en `player_y`: positie van de speler (een blauw blokje onderaan)
- `rock_x` en `rock_y`: positie van de steen

- `screen.draw.filled_rect(...)`: tekent een blokje op het scherm

?? Opdracht

- Verplaats de steen naar een andere plek op het scherm door `rock_x` en `rock_y` aan te passen
- Maak de speler breder of smaller
- Verander de kleuren van speler en steen

?? Extra uitdaging

- Teken meerdere stenen op het scherm (gebruik meerdere `draw.filled_rect()`)

? Inleveren

1. Maak een screenshot waarop je de speler en minstens één steen ziet (waarbij je de plaats van de steen dus hebt aangepast).

2 Speler bewegen

In deze les gaan we de speler besturen met de pijltjestoetsen.

De speler beweegt alleen naar links en rechts, en mag niet buiten het scherm gaan.

Wat gaan we doen?

We bewerken de `update()`-functie om de `x`-positie van de speler aan te passen als je op pijltjes drukt.

image.png

We voegen een maximale en minimale positie toe zodat de speler niet van het scherm glijdt.

? Code

```
import pgzrun
```

```
WIDTH = 800
HEIGHT = 600

player_x = 400
player_y = 550
player_width = 80
player_height = 20
player_speed = 5

rock_x = 300
rock_y = 0
rock_size = 40

def draw():
    screen.clear()
    screen.draw.filled_rect(Rect((player_x, player_y), (player_width, player_height)), "blue")
    screen.draw.filled_rect(Rect((rock_x, rock_y), (rock_size, rock_size)), "gray")

def update():
    global player_x

    if keyboard.left:
        player_x -= player_speed
    if keyboard.right:
        player_x += player_speed

    # Speler binnen scherm houden
    if player_x < 0:
        player_x = 30
    if player_x > WIDTH - player_width:
        player_x = WIDTH - player_width

pgzrun.go()
```

?? Uitleg

- `player_speed`: hoe snel de speler beweegt
- `keyboard.left` en `keyboard.right`: detecteren of een toets is ingedrukt

- `if player_x > WIDTH - player_width`: voorkomt dat de speler buiten beeld schuift

?? Opdracht

- Beweeg de speler heen en weer met je pijltjestoetsen
- Verander `player_speed` – wordt de speler sneller of trager?
- Aan de linker kant van het scherm stuitert de speler terwijl aan de rechterkant de speler netjes op de rand stopt. Zie jij hoe dat komt? Probeer dit aan te passen, probeer gewoon maar wat, je kan niets kapot maken!

> Tip: bij welke `if` wordt gekeken of de speler tegen de linkerkant van het scherm aan zit?

◫ Extra uitdaging

- Laat de speler sneller bewegen als je de toets langer inhoudt
- Laat de speler automatisch naar links of rechts bewegen als je een extra toets indrukt (bijvoorbeeld `A` of `D`)

? Inleveren

1. Leg in eigen woorden uit hoe de `player_speed` de snelheid van het spel veranderd.
2. Leg uit waarom de speler aan de linkerkant van het scherm 'stuitert' en wat heb je aangepast om dit te voorkomen?

3 Steen laten vallen

In deze les gaan we de steen automatisch laten vallen.

Zodra de steen de onderkant van het scherm bereikt, verschijnt hij opnieuw bovenaan op een willekeurige plek.

image.png

Wat gaan we doen?

We voegen een `rock_speed` toe en laten de `y`-positie van de steen langzaam toenemen in `update()`.

image.png

We controleren of de steen het scherm uit valt, en zetten hem dan opnieuw bovenaan met een willekeurige x-positie.

? Code

```
import pgzrun
import random

WIDTH = 800
HEIGHT = 600

player_x = 400
player_y = 550
player_width = 80
player_height = 20
player_speed = 5

rock_x = random.randint(0, WIDTH - 40)
rock_y = 0
rock_size = 40
rock_speed = 3

def draw():
    screen.clear()
    screen.draw.filled_rect(Rect((player_x, player_y), (player_width, player_height)), "blue")
    screen.draw.filled_rect(Rect((rock_x, rock_y), (rock_size, rock_size)), "gray")

def update():
    global player_x, rock_y, rock_x, rock_size

    if keyboard.left:
        player_x -= player_speed
    if keyboard.right:
        player_x += player_speed

    if player_x < 0:
        player_x = 0
    if player_x > WIDTH - player_width:
```

```
        player_x = WIDTH - player_width

    # Steen laten vallen
    rock_y += rock_speed

    # Als de steen onderaan is, zet hem weer bovenaan met random x
    if rock_y > HEIGHT:
        rock_y = 0
        rock_x = random.randint(0, WIDTH - rock_size)

pgzrun.go()
```

?? Uitleg

- `rock_speed`: bepaalt hoe snel de steen valt
- `rock_y += rock_speed`: laat de steen naar beneden bewegen
- `random.randint(...)`: zorgt voor een willekeurige x-positie als de steen opnieuw verschijnt

?? Opdracht

- Verander de `rock_speed` – wat gebeurt er?
- Maak de steen groter of kleiner door `rock_size` aan te passen.
- Zorg er nu voor dat elke keer als de steen opnieuw valt de rock_size wordt aangepast en een random grootte krijgt.
- Met random.randint(1, 10) wordt er een getal tussen 1 en 10 gegenereerd. Bedenk zelf mooie waarden en pas de code aan zodat de grootte van de steen telkens anders wordt.

□□ Extra uitdaging

- Laat meerdere stenen tegelijk vallen
- Laat elke steen een willekeurige snelheid hebben

? Inleveren

1. Leg uit hoe je ervoor hebt gezorgdt dat de steen telkens een ander grootte kijgt.

4 Botsing & Game Over

In deze les gaan we controleren of de speler de steen raakt.

image.png

Als er een botsing is, stopt het spel en verschijnt er de tekst “Game Over”.

Wat gaan we doen?

We maken een `Rect` van de speler en van de steen, en gebruiken `colliderect()` om te zien of ze elkaar raken.

We gebruiken een variabele `game_over` om te stoppen met het spel als er een botsing is.

? Code

```
import pgzrun
import random
from pygame import Rect

WIDTH = 800
HEIGHT = 600

player_x = 400
player_y = 550
player_width = 80
player_height = 20
player_speed = 5

rock_x = random.randint(0, WIDTH - 40)
rock_y = 0
rock_size = 40
rock_speed = 3

game_over = False
```

```
def draw():
    screen.clear()
    if game_over:
        screen.draw.text("GAME OVER", center=(WIDTH // 2, HEIGHT // 2), fontsize=60,
color="red")
        return

    screen.draw.filled_rect(Rect((player_x, player_y), (player_width, player_height)), "blue")
    screen.draw.filled_rect(Rect((rock_x, rock_y), (rock_size, rock_size)), "gray")

def update():
    global player_x, rock_y, rock_x, game_over

    if game_over:
        return

    if keyboard.left:
        player_x -= player_speed
    if keyboard.right:
        player_x += player_speed

    if player_x < 0:
        player_x = 0
    if player_x > WIDTH - player_width:
        player_x = WIDTH - player_width

    rock_y += rock_speed

    if rock_y > HEIGHT:
        rock_y = 0
        rock_x = random.randint(0, WIDTH - rock_size)

    # Botsing detecteren
    speler_rect = Rect(player_x, player_y, player_width, player_height)
    steen_rect = Rect(rock_x, rock_y, rock_size, rock_size)

    if speler_rect.colliderect(steen_rect):
        game_over = True
```

```
pgzrun.go()
```

?? Uitleg

- `Rect(x, y, w, h)` maakt een rechthoek op de juiste plek
- `colliderect()` kijkt of twee rechthoeken elkaar raken
- `game_over` bepaalt of het spel nog doorgaat

?? Opdracht

- Laat de speler de steen raken en kijk of het spel stopt
- Verplaats de speler naar de andere kant van het scherm – bots je dan nog?
- Verander de “GAME OVER” tekst (bijvoorbeeld kleur of grootte)

?? Extra uitdaging

- Laat het spel herstarten als je op de `R`-toets drukt
- Speel een geluid af bij de botsing (bijv. `sounds.hit.play()`)

? Inleveren

1. Leg eersts stap-voor-stap in he eigen woorden uit hoe er een botsing van de steen met de speler wordt gedetecteerd.
2. Leg daarna stap-voor-stap, in eigen woorden uit wat er allemaal gebeurt als de steen tegen de speler aan komt. Verwijs daarbij naar de code.

5 Meerdere stenen & moeilijker maken

In deze les gaan we meerdere stenen tegelijk laten vallen.

Bovendien maakt het spel zichzelf moeilijker naarmate je langer speelt: de stenen vallen sneller.

Wat gaan we doen?

We maken een lijst van stenen, elk met hun eigen positie en snelheid.

image.png

We laten deze stenen vallen en bij een botsing stoppen we het spel.

We laten het spel steeds moeilijker worden door de snelheid te verhogen na een paar seconden.

? Code

```
import pgzrun
import random
from pygame import Rect

WIDTH = 800
HEIGHT = 600

player_x = 400
player_y = 550
player_width = 80
player_height = 20
player_speed = 5

rocks = []
game_over = False
rock_timer = 0
rock_interval = 60  # frames
difficulty = 1.0

# Start met een paar stenen
for _ in range(3):
    x = random.randint(0, WIDTH - 40)
    speed = random.uniform(2, 4)
    rocks.append({"x": x, "y": 0, "size": 40, "speed": speed})

def draw():
    screen.clear()
    if game_over:
        screen.draw.text("GAME OVER", center=(WIDTH // 2, HEIGHT // 2), fontsize=60,
color="red")
        return
```

```
    screen.draw.filled_rect(Rect((player_x, player_y), (player_width, player_height)), "blue")
    for rock in rocks:
        screen.draw.filled_rect(Rect((rock["x"], rock["y"]), (rock["size"], rock["size"])),
"gray")

def update():
    global player_x, game_over, rock_timer, difficulty

    if game_over:
        return

    if keyboard.left:
        player_x -= player_speed
    if keyboard.right:
        player_x += player_speed

    player_x = max(0, min(WIDTH - player_width, player_x))

    speler_rect = Rect(player_x, player_y, player_width, player_height)

    for rock in rocks:
        rock["y"] += rock["speed"] * difficulty

        if rock["y"] > HEIGHT:
            rock["y"] = 0
            rock["x"] = random.randint(0, WIDTH - rock["size"])
            rock["speed"] = random.uniform(2, 5)

        rock_rect = Rect(rock["x"], rock["y"], rock["size"], rock["size"])
        if speler_rect.colliderect(rock_rect):
            game_over = True

    # Verhoog de moeilijkheid langzaam
    rock_timer += 1
    if rock_timer % 300 == 0:
        difficulty += 0.2

pgzrun.go()
```

?? Uitleg

- Elke steen is een dict met `x`, `y`, `size` en `speed`
- We gebruiken een `for`-loop om alle stenen te laten vallen
- Met `difficulty` verhogen we langzaam de snelheid van de stenen

  image.png

?? Opdracht

- Test het spel en kijk of het spel moeilijker wordt na ongeveer 15 seconden!
- Nu gaan we de grootte van de stenen weer aanpassen naar een random waarde zoals we dat bij de vorige opdracht ook hebben gedaan.

  Zorg er dus voor dat bij het aanmaken van de stenen iedere steen een andere (random) grootte krijgt.

  Dat gaat iets anders omdat we nu meerdere 'rocks' hebben. In de `for rock in rocks:` loop worden één voor één alle blokken behandeld. De grootte van de rock staat in `rock["size"]`

> Tip: weet je nog dat je met `random.randint(1,4)` een getal tussen 1 en 4 kan genereren?

□□ Extra uitdaging

- Laat het aantal stenen toenemen naarmate het spel langer duurt
- Laat een andere kleur steen verschijnen bij hogere moeilijkheid.
- Laat elke steen een ander formaat hebben
- Toon je “score” op het scherm: hoe lang heb je overleefd?

? Inleveren

1. Lever je code (.py bestand) in.

6 Score, Reflectie & Uitbreiding

In deze les voeg je een score toe die laat zien hoelang je het hebt volgehouden.

Daarnaast kijk je terug op je eigen werk én kies je een uitbreiding om het spel leuker of moeilijker te maken.

Game flow

image.png

Wat gaan we doen?

- We voegen een `score` toe die telt hoeveel frames je overleeft
- We tonen deze score linksboven op het scherm
- Bij Game Over laten we je eindscore zien

? Toevoegingen aan bestaande code

```
score = 0  # bij de andere variabelen

def draw():
    screen.clear()
    if game_over:
        screen.draw.text("GAME OVER", center=(WIDTH // 2, HEIGHT // 2), fontsize=60,
color="red")
        screen.draw.text(f"Score: {score}", center=(WIDTH // 2, HEIGHT // 2 + 50),
fontsize=40, color="black")
        return
    ...
    screen.draw.text(f"Score: {score}", (10, 10), fontsize=40, color="black")

def update():
    global score
    if game_over:
        return
    ...
    score += 1
```

?? Uitleg

- `score` wordt bij elke frame 1 hoger → hoe langer je speelt, hoe hoger je score
- `screen.draw.text()` toont je score tijdens én na het spel

? Reflectie

Beantwoord de volgende vragen onderaan je code als commentaar of in een apart document:

- Wat vond je het leukst om te maken in dit spel?
- Wat vond je moeilijk, en hoe heb je dat opgelost?
- Wat heb je geleerd over Python en programmeren?
- Waar ben je trots op?

? Uitbreiding (kies er één)

- □□ Laat een explosie zien of geluid horen bij Game Over
- □□ Voeg bewegende obstakels toe
- □□ Houd een highscore bij (hoogste score van de sessie)
- □□ Geef de speler power-ups: tijdelijk onsterfelijk, versnellen, etc.
- □□ Verander het uiterlijk van de speler of achtergrond na elke 30 seconden

□□ Eigen idee?

Bedenk zelf een uitbreiding en probeer deze te maken. Schrijf kort op wat jouw idee is en hoe je het hebt aangepakt.

? Inleveren

- Lever je eindversie van het spel in (inclusief score en uitbreiding)
- Voeg een korte beschrijving toe van wat je hebt toegevoegd of aangepast
- Optioneel: voeg een screenshot of kort filmpje toe van je spel

7 Reflectieopdracht

Deze reflectieopdracht helpt je om stil te staan bij wat je hebt geleerd tijdens het programmeren van je spel.

Beantwoord de onderstaande vragen eerlijk en in je eigen woorden. Je mag je antwoorden inleveren via een apart document of onderaan je Python-bestand als commentaar zetten.

? Vragen

- Wat vond je het leukste om te maken of uit te proberen?
- Wat vond je lastig? Hoe heb je dat opgelost?
- Wat heb je geleerd over Python (of programmeren in het algemeen)?
- Waar ben je trots op in jouw eindresultaat?
- Wat zou je de volgende keer anders willen doen of verbeteren?

? Inleveren

Lever je reflectie in bij je docent via de afgesproken manier (document, upload of als commentaar in je code).

###

 # Docenten

? Docentenhandleiding

? Overzicht

- **Doelgroep:** Leerlingen met basiskennis Python (13-16 jaar)
- **Duur:** 6 lessen van ±45-60 minuten
- **Software:** Thonny + Pygame Zero
- **Spelconcept:** De speler ontwijkt vallende objecten (stenen) en het spel wordt steeds moeilijker

? Leerdoelen

- Werken met coördinaten en schermlogica
- Gebruik van `draw()` en `update()`
- Toetsenbordbesturing met `keyboard.left` en `keyboard.right`
- Beweging simuleren met variabelen
- Objecten detecteren met `Rect` en `colliderect()`
- Werken met lijsten voor meerdere objecten
- Reflecteren op het leerproces en zelf uitbreidingen bedenken

? Lesoverzicht

Les	Onderwerp	Nieuwe concepten
1	Speler en steen tekenen	`draw()`, rechthoek tekenen, coördinaten
2	Speler beweegt	`update()`, keyboard input, begrenzen
3	Steen valt en komt terug	Beweging, `random`, logica
4	Botsing + Game Over	`Rect`, `colliderect()`, boolean vlag
5	Meerdere stenen + moeilijkheid	Lijsten, `for`-loops, moeilijkheidsschaal
6	Score, Reflectie & Uitbreiding	Scorevariabele, `draw.text()`, vrije opdracht

?? Valkuilen

- Speler glijdt van het scherm → `if player_x > WIDTH - breedte` vergeten
- Botsing werkt niet → verkeerde waarden in `Rect()`
- Alle stenen bewegen tegelijk, maar maar één wordt gecheckt op botsing
- Score telt door na game over → geen check op `if not game_over:`

? Differentiatie

Voor snelle leerlingen

- Voeg power-ups toe (onzichtbaarheid, levens, vertraging)
- Laat speler en stenen met sprites tekenen in plaats van rechthoeken
- Laat leerlingen zelf een nieuw spelelement verzinnen

Voor langzamere leerlingen

- Laat maar één steen vallen (geen lijst)
- Geef per les kant-en-klare startercode mee
- Werk samen in tweetallen

? Beoordeling (optioneel)

Criterium	Omschrijving	Score (1-5)
Werkt het spel	Geen fouten, spel werkt zoals bedoeld	□□
Code is leesbaar	Logische structuur, goede naamgeving	□□
Uitbreiding toegevoegd	Creatieve of technische aanvulling	□□
Reflectie	Antwoorden zijn volledig en met inzicht	□□

? Tips voor in de les

- Laat leerlingen na elke wijziging op F5 drukken → directe feedback is motiverend
- Gebruik klassikale demo's bij fouten: "Waarom crasht deze versie?"
- Laat leerlingen zelfstandig kleine uitbreidingen testen vanaf les 5
- Bespreek reflectievragen klassikaal in les 6

? Benodigdheden

- Thonny geïnstalleerd (met Pygame Zero via `pip install pgzero` indien nodig)
- Werkende computer (Windows, Linux of macOS)
- Toetsenbord met pijltjestoetsen

Snake

0 Snake Lessenserie

datasource

Welkom bij de Snake lessenserie! In deze serie leer je stap voor stap hoe je het klassieke spelletje **Snake** maakt met `Python` en `Pygame Zero`. Je begint met het tekenen van een blokje en eindigt met een compleet werkend spel – inclusief groeiende slang, score en Game Over!

? Overzicht van de lessen

1. Les 1 – Teken de slangkop
2. Les 2 – Beweeg de slang
3. Les 3 – Botsing met schermrand
4. Les 4 – Richting automatisch volgen
5. Les 5 – Teken een appel en detecteer botsing
6. Les 6 – Laat de slang groeien
7. Les 7 – Begrijp insert() en pop()
8. Les 8 – Vertraagde beweging
9. Les 9 – Score en Game Over
10. Les 10 – Challenge en uitbreiding

? Leerdoelen

- Je leert werken met de `draw()` en `update()` functies van Pygame Zero
- Je oefent met variabelen, lists en toetsenbordinvoer
- Je leert wat een game loop is en hoe je controle krijgt over beweging

- Je leert je eigen code uitbreiden, testen en verbeteren

?? Benodigdheden

- Thonny geïnstalleerd met de package `pgzero`
- Een beetje basiskennis van Python (variabelen, if, functies)

? Hoe werk je?

Elke les bevat uitleg, voorbeeldcode, opdrachten én een extra uitdaging. Voer je code steeds uit in Thonny en probeer alle opdrachten echt zelf op te lossen. Je mag hulp vragen of AI gebruiken, maar probeer te begrijpen wat er gebeurt.

? Klaar?

Lever je eindspel in, samen met je antwoorden op de reflectievraag. Veel succes en vooral: veel plezier met programmeren!

?? Opdracht

We gaan nog een projectje maken met Thonny (uit de vorige les) en bij dit project gaan we gebruik maken van de standaard Python library:

pgzero

Weet je nog hoe je een package installeert in Thonny?

Yep, Tools - Manage Packages en dan pgzero zoeken en installeren.

? Inleveren

Maak een neiuw bestand en bewaar dit lege bestand op je computer. Maak een screenshot van Thonny met een leeg bestand.

> Tip: Kopieer je vorige project en noem het anders.

Voorbeeld

image.png

1 Teken de slangkop

In deze les gaan we beginnen met het maken van een eigen versie van Snake in Python met Pygame Zero.

We gaan eerst de kop van de slang tekenen op het scherm. Je ziet dan een vierkantje dat straks kan gaan bewegen.

Wat gaan we doen?

We maken een slangkop als vierkant met een bepaalde positie en grootte, en tekenen die in de `draw()`-functie.

? Benodigdheden

- Een werkende Python-omgeving met Pygame Zero (zoals Thonny)
- Geen plaatjes nodig – we tekenen de slang met blokken

? Startercode

```
# importeer library
import pgzrun

# Spelgrootte
WIDTH = 600
HEIGHT = 400

# Startpositie van de slangkop
snake_x = 100
snake_y = 100
tile_size = 20

def draw():
    screen.clear()
    screen.draw.filled_rect(Rect((snake_x, snake_y), (tile_size, tile_size)), "green")

#start programma
pgzrun.go()
```

?? Uitleg

- `WIDTH` en `HEIGHT`: grootte van het spelvenster
- `snake_x` en `snake_y`: positie van de slangkop
- `tile_size`: grootte van het blokje
- `screen.draw.filled_rect(...)`: tekent een gevuld vierkantje op het scherm

?? Opdracht

- Pas de waarde van `snake_x` en `snake_y` aan – wat gebeurt er?
- Maak het vierkant groter of kleiner door `tile_size` te wijzigen
- Verander de kleur van het vierkant in bijvoorbeeld `"blue"` of `"orange"`

⬜⬜ Extra uitdaging

Teken een tweede blokje naast de slangkop alsof er al een stukje staart is. Gebruik nog een `screen.draw.filled_rect()`.

? Inleveren

1. Maak een screenshot van je 'slangkop' op het scherm (dus geen code) waarbij je de 'slangkop'duidelijke op een andere positie hebt gezet (bijvoorbeeld rechtsonder in het schrem, of in het midden).

2 Beweeg de slang

In deze les gaan we de slangkop laten bewegen met de pijltjestoets die je indrukt.

We gebruiken daarvoor de `update()`-functie van Pygame Zero en de toetsenbordinput.

Wat gaan we doen?

We maken een richting-variabele en passen de positie van de slang aan op basis van de pijltjes die je indrukt.

? Code

Dit is een **deel van de code**, je moet jouw bestaande code aanpassen aan de hand van deze nieuwe code.

Je hoeft dus niet alles opnieuw te typen. Plaats de `update()` functie en pas eventueel `snake_x`, `snake_y`, `tile_size` en `step` aan in jouw bestaande code.

```
# Startpositie van de slangkop
snake_x = 200
snake_y = 200
tile_size = 10
step = 1

def draw():
    screen.clear()
    screen.draw.filled_rect(Rect((snake_x, snake_y), (tile_size, tile_size)), "green")

def update():
    global snake_x, snake_y

    if keyboard.left:
        snake_x -= step
    if keyboard.right:
        snake_x += step
    if keyboard.up:
        snake_y -= step
    if keyboard.down:
        snake_y += step
```

?? Uitleg

- `update()` wordt automatisch meerdere keren per seconde uitgevoerd
- `keyboard.left` controleert of de linkerpijl is ingedrukt
- Telkens als je een toets indrukt, verandert de positie van de slang
- `step` bepaalt hoe ver de slang per stap beweegt

?? Opdracht

- Beweeg de slang door het scherm met de pijltjes

- Pas `step` aan naar een andere waarde – wat merk je?
- Laat de slang sneller of langzamer bewegen door minder of meer pixels per keer te verplaatsen

◫ Extra uitdaging

Laat de slang automatisch blijven bewegen in de laatst gekozen richting:

- Gebruik een variabele `richting` die je bijwerkt met `on_key_down()`
- Laat de slang dan elke update in die richting verder bewegen

? Inleveren

1. Leg uit wat de variable step doet.
2. Welke waarde heb je gekozen, waarom?

(je kunt dit inleveren in het tekst veld of in een .txt. bestandje)

3 Automatische beweging

In deze les gaan we de slang automatisch laten blijven bewegen in de richting van de laatste pijltjestoets die je hebt ingedrukt.

? Wat gaan we doen?

- We maken een nieuwe variabele `richting`.
- Als je op een pijltjestoets drukt, verandert de waarde van `richting`.
- In de `update()` functie verplaatst de slang zich elke keer opnieuw in de gekozen richting – ook als je de toets niet ingedrukt houdt!

? Code

Gebruik deze versie als nieuwe code (je mag dit toevoegen aan of combineren met je bestaande code):

```
import pgzrun

WIDTH = 600
HEIGHT = 400

snake_x = 200
snake_y = 200
tile_size = 20
step = 10
richting = "right"

def draw():
    screen.clear()
    screen.draw.filled_rect(Rect((snake_x, snake_y), (tile_size, tile_size)), "green")

def update():
    global snake_x, snake_y, richting

    if richting == "left":
        snake_x -= step
    elif richting == "right":
        snake_x += step
    elif richting == "up":
        snake_y -= step
    elif richting == "down":
        snake_y += step

def on_key_down(key):
    global richting

    if key == keys.LEFT:
        richting = "left"
    elif key == keys.RIGHT:
        richting = "right"
    elif key == keys.UP:
        richting = "up"
    elif key == keys.DOWN:
        richting = "down"
```

```
pgzrun.go()
```

?? Uitleg

- `richting`: deze variabele onthoudt de laatst gekozen richting
- `on_key_down()`: dit is een functie die wordt uitgevoerd als je op een toets drukt
- `update()`: deze functie verplaatst de slang elke keer in de gekozen richting, ook als je geen toets indrukt

?? Opdracht

- Test of je slang automatisch blijft bewegen nadat je een pijl indrukt
- Wat gebeurt er als je twee keer snel op een andere richting drukt?
- Probeer met `tile_size` en `step` te spelen voor een ander effect

□□ Extra uitdaging

Voeg grenzen toe aan je spel: laat de slang stoppen of ergens anders naartoe gaan als hij de rand van het scherm raakt.

? Inleveren

1. Leg in je eigen woorden uit wat de functie `on_key_down()` doet

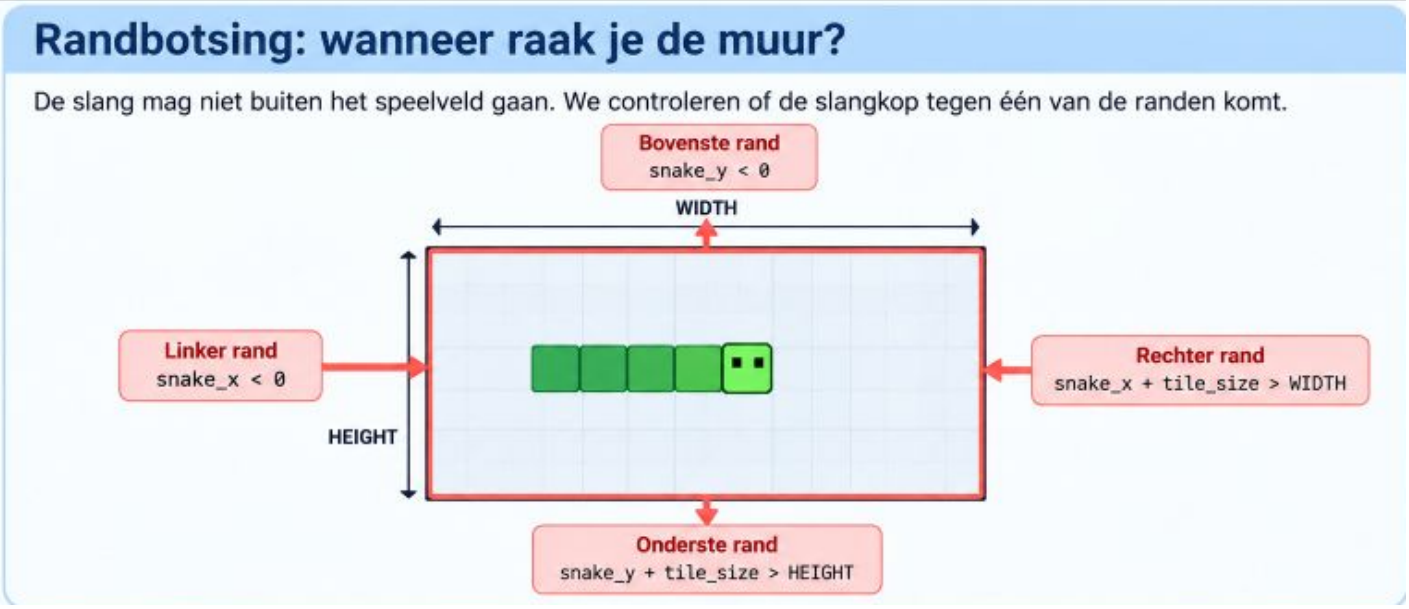
(Lever dit in via het tekstvak of via een upload.)

4 Randbotsing en automatische richting

In deze les zorgen we ervoor dat de slang niet zomaar het scherm verlaat. Zodra hij een rand raakt, verandert hij automatisch van richting. Zo beweegt hij steeds binnen het scherm!

? Wat gaan we doen?

- We controleren of de slang de rand van het scherm raakt.
- Als dat zo is, passen we automatisch de richting aan.



`snake_x` en `snake_y` geven de positie van de **linkerbovenhoek van de slangkop** weer. De kop zelf heeft ook een breedte en hoogte van `tile_size`.

? Code (let op: onvolledig!)

Hieronder zie je de code. De code is onvolledig en moet worden aangepast.

> Hint: kijk naar de ... (drie puntjes)

```
import pgzrun

WIDTH = 600
HEIGHT = 400

snake_x = 200
snake_y = 200
tile_size = 20
step = 5
richting = "right"

def draw():
    screen.clear()
    screen.draw.filled_rect(Rect((snake_x, snake_y), (tile_size, tile_size)), "green")

def update():
    global snake_x, snake_y, richting
```

```
    if richting == "left":
        snake_x -= step
    elif richting == "right":
        snake_x += step
    elif richting == "...":
        snake_y -= step
    elif richting == "...":
        snake_y += step

    # □□Hier controleren we of de slang de rand raakt
    if snake_x < 0:
        richting = "right"
    elif snake_x + tile_size > WIDTH:
        richting = "left"
    elif snake_y < 0:
        richting = "..."
    elif snake_y + tile_size > HEIGHT:
        richting = "..."

def on_key_down(key):
    global richting

    if key == keys.LEFT:
        richting = "left"
    elif key == keys.RIGHT:
        richting = "right"
    elif key == keys.UP:
        richting = "up"
    elif key == keys.DOWN:
        richting = "down"

pgzrun.go()
```

Als je de linker- of rechterkant raakt, dan 'stuitert' je terug. Dat gebeurt niet als je de boven- of onderkant aanraakt. Pas de code aan zodat je ook naar boven en beneden kan bewegen en zodat je ook terugstuiters als je de onder-of bovenkant aanraakt.

?? Uitleg

- `snake_x + tile_size > WIDTH`: betekent dat de slang voorbij de rechterrand gaat
- Als de slang een rand raakt, verander je de variabele `richting`

?? Opdracht

- Pas de code aan zodat de slang alle richtingen uit kan bewegen en zodat hij als hij de rand raakt (link, recht en boven en beneden) terug 'stuiterd'.
- Test of de slang de alle richtingen op gaat en hij aan alle kantne goed terug 'stuiterd'!

◫ Extra uitdaging

Laat de slang bij elke randbotsing van kleur veranderen! (Tip: maak een variabele `kleur` en gebruik bijvoorbeeld `random.choice()`)

? Inleveren

1. Leg in **eigen woorden** uit **wat** je hebt aangepast op regel 35 en 36 en **hoe dat werkt.**

Vind je dit lastig? Dan *kun* je de deze template gebruiken:

```
Ik heb op regel 35 .... aangepast en ik heb op regel 36 .... aangepast.
Wat er dan gebeurt is dat als de Snake ......... en de toets ..... wordt ingedrukt dan
verandert de .......... van .......... naar ............
```

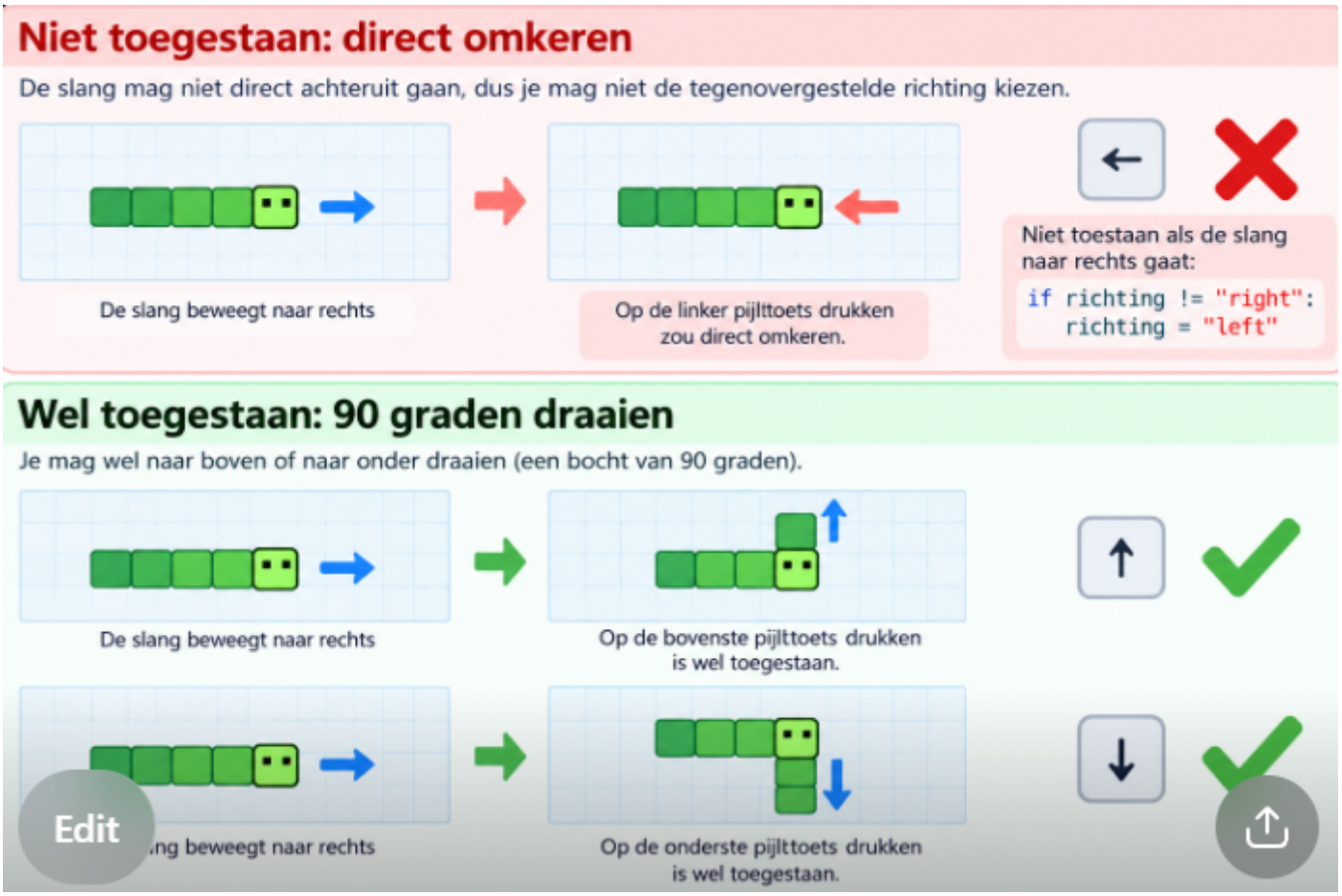
(Lever dit in via het tekstvak of via een upload.)

5 Geen omkeren toegestaan

In deze les zorgen we ervoor dat de slang niet meteen omkeert. In een echte Snake-game kun je namelijk niet ineens van rechts naar links bewegen – dan zou de slang zichzelf opeten!

? Wat gaan we doen?

- We voorkomen dat de slang direct de tegenovergestelde richting kiest.
- We vergelijken de huidige richting met de nieuwe richting voordat we die veranderen.



? Code (let op: onvolledig!)

De volgende code voorkomt omkeren, maar jij moet één voorwaarde nog zelf aanvullen.

```
import pgzrun

WIDTH = 600
HEIGHT = 400

snake_x = 200
snake_y = 200
tile_size = 20
step = 1
richting = "right"

def draw():
    screen.clear()
    screen.draw.filled_rect(Rect((snake_x, snake_y), (tile_size, tile_size)), "green")

def update():
```

```
    global snake_x, snake_y, richting

    if richting == "left":
        snake_x -= step
    elif richting == "right":
        snake_x += step
    elif richting == "up":
        snake_y -= step
    elif richting == "down":
        snake_y += step

def on_key_down(key):
    global richting

    if key == keys.LEFT and richting != "right":
        richting = "left"
    elif key == keys.RIGHT and richting != "left":
        richting = "right"
    elif key == keys.UP and richting != "down":
        richting = "up"
    elif key == keys.DOWN and richting != "...":
        richting = "down"

pgzrun.go()
```

Stuiteren tegen randen werkt niet meer

Het botsen tegen de randen wat in stap 4 is gedaan, zit niet in deze code. Dat is express gedaan. We proberen op deze manier een ding gelijktijdig uit te leggen en het voorkomt dat je op dit moment "*door de bomen het bos niet meer ziet*".

?? Uitleg

- We gebruiken een `if`-voorwaarde om te voorkomen dat je teruggaat in de tegenovergestelde richting.
- De ontbrekende regel is die voor `keys.DOWN`: welke richting is dan niet toegestaan?

?? Opdracht

- Vul de ontbrekende voorwaarde aan zodat de slang niet van "up" naar "down" mag keren
- Test alle richtingen: kun je nog steeds normaal draaien? Keren lukt niet meer, toch?

◫ Extra uitdaging

Laat de slang een geluidje maken als je op een toets drukt, maar de richting wordt niet veranderd (bijv. als je wél op links drukt, maar dat mag niet).

Met deze code kan je ene geluidje afspelen.

```
sounds.beep.play()
```

Om een sound af te kunnen spelen moet jouw project folder een mapje maken sounds en daarin moet een beep.wav komen.

```
mijn_project/
├── main.py
└── sounds/
    └── beep.wav
```

Geluidjes toevoegen

Je kunt zelf geluidjes toevoegen, stel je hebt een geluidje **bel.wav** dan plaats je dat in de sounds directory en dan gebruik je het commando

`souds.bel.play()`

? Inleveren

1. Leg uit **wat** je hebt veranderd en **hoe** het werkt (dit zijn dus twee vragen: **wat** en **hoe**).

Vind je dit lastig? Dan *kun* je de deze template gebruiken:

```
Ik heb op regel ...     ..... aangepast.
Wat er dan gebeurt is dat als de Snake ......... en de toets ..... wordt ingedrukt dan
.........
```

(Lever dit in via het tekstvak of via een upload.)

6 Appel tekenen en raken

In deze les voegen we een appel toe aan het spel. De appel verschijnt op een willekeurige plek op het scherm. Als de slang de appel raakt, verschijnt er een bericht in de console.

? Wat gaan we doen?

- We gebruiken de `random`-module om een appel op een willekeurige positie te tekenen.
- We tekenen de 'appel' met een geel cirkeltje of vierkantje.
- We detecteren of de slang de appel raakt.

? Benodigdheden

- Je slangkop beweegt al automatisch over het scherm
- Je hebt al gewerkt met `tile_size` en `snake_x`/`snake_y`

? Code (let op: onvolledig!)

Onderstaande code tekent een slangkop en een appel. De botsing werkt al, maar na een botsing blijft de appel op dezelfde plek staan. Vul zelf aan wat er moet gebeuren!

```
import pgzrun
import random

WIDTH = 600
HEIGHT = 400

tile_size = 20
snake_x = 100
snake_y = 100
richting = "right"
step = 3

# Appelpositie (willekeurig op het grid)
apple_x = random.randint(0, (WIDTH - tile_size) // tile_size) * tile_size
apple_y = random.randint(0, (HEIGHT - tile_size) // tile_size) * tile_size
```

```
def draw():
    screen.clear()
    screen.draw.filled_rect(Rect((snake_x, snake_y), (tile_size, tile_size)), "green")
    screen.draw.filled_circle((apple_x + tile_size // 2, apple_y + tile_size // 2), tile_size
// 2, "yellow")

def on_key_down(key):
    global richting

    if key == keys.LEFT and richting != "right":
        richting = "left"
    elif key == keys.RIGHT and richting != "left":
        richting = "right"
    elif key == keys.UP and richting != "down":
        richting = "up"
    elif key == keys.DOWN and richting != "up":
        richting = "down"

def update():
    global snake_x, snake_y, apple_x, apple_y, richting

    if richting == "left":
        snake_x -= step
    elif richting == "right":
        snake_x += step
    elif richting == "up":
        snake_y -= step
    elif richting == "down":
        snake_y += step

    # Botst de slang met de appel?
    snake_rect = Rect((snake_x, snake_y), (tile_size, tile_size))
    apple_rect = Rect((apple_x, apple_y), (tile_size, tile_size))
    if snake_rect.colliderect(apple_rect):
        print("□□Appel geraakt!")
        # □□VUL HIER AAN: genereer een nieuwe positie voor de appel
        # apple_x = ...
        # apple_y = ...
```

```
pgzrun.go()
```

?? Uitleg

- `random.randint()`: geeft een willekeurig geheel getal terug
- De appelpositie wordt op het grid berekend, zodat deze altijd netjes uitlijnt
- `filled_circle()`: tekent een geel rondje (je kunt ook `filled_rect()` gebruiken)

?? Opdracht

- Test of de appel op het scherm verschijnt
- Beweeg de slang naar de appel – zie je de console-uitvoer?
- Vul de ontbrekende regels in zodat de appel na een botsing op een **nieuwe plek** verschijnt

◻◻ Extra uitdaging

Laat de appel niet precies op dezelfde plek als de slang verschijnen wanneer hij opnieuw wordt gegenereerd!

? Inleveren

1. Lever de code in .py bestand.

7 De slang groeit

In deze les maak je van je slang een échte slang: eentje die uit meerdere blokjes bestaat en langer wordt als hij een appel eet.

? Wat gaan we doen?

- We veranderen de slang van één blokje naar een lijst van blokjes
- De slang groeit bij het eten van een appel
- We houden de lengte gelijk als er geen appel wordt gegeten

? Strategie

De code wordt nu iets ingewikkelder. Het is niet erg als je niet alles direct begrijpt. Probeer wel te volgen wat er gebeurt. We bespreken de strategie:

1. De slang bestaat uit meerdere delen. De code in de `draw()`-functie tekent elk stukje.
2. Bij elke update wordt er een nieuw blokje toegevoegd aan de kop van de slang.
3. Als de slang een appel raakt: dan blijft de slang langer.
4. Als er geen appel is geraakt: dan wordt het laatste stukje van de slang verwijderd.

Dus in het kort:

1. Teken de slang
2. Voeg een stukje toe aan de kop
3. Geen appel? Verwijder de staart

ChatGPT Image 3 Oct 2026, 18_04_34.jpg

? Code

```
import pgzrun
import random

WIDTH = 600
HEIGHT = 400
tile_size = 20

snake = [(100, 100)]
richting = "right"
step = 3

apple_x = random.randint(0, (WIDTH - tile_size) // tile_size) * tile_size
apple_y = random.randint(0, (HEIGHT - tile_size) // tile_size) * tile_size
```

```
def draw():
    screen.clear()
    for segment in snake:
        screen.draw.filled_rect(Rect(segment, (tile_size, tile_size)), "green")
    screen.draw.filled_circle((apple_x + tile_size // 2, apple_y + tile_size // 2), tile_size
// 2, "yellow")

def on_key_down(key):
    global richting
    if key == keys.LEFT and richting != "right":
        richting = "left"
    elif key == keys.RIGHT and richting != "left":
        richting = "right"
    elif key == keys.UP and richting != "down":
        richting = "up"
    elif key == keys.DOWN and richting != "up":
        richting = "down"

def update():
    global apple_x, apple_y, richting

    head_x, head_y = snake[0]
    if richting == "left":
        head_x -= step
    elif richting == "right":
        head_x += step
    elif richting == "up":
        head_y -= step
    elif richting == "down":
        head_y += step

    new_head = (head_x, head_y)
    snake.insert(0, new_head)

    head_rect = Rect(new_head, (tile_size, tile_size))
    apple_rect = Rect((apple_x, apple_y), (tile_size, tile_size))

    if head_rect.colliderect(apple_rect):
        print("□□Appel geraakt!")
```

```
        apple_x = random.randint(0, (WIDTH - tile_size) // tile_size) * tile_size
        apple_y = random.randint(0, (HEIGHT - tile_size) // tile_size) * tile_size
    else:
        snake.pop()  # staart verwijderen

    if new_head in snake[1:]: # Deze code detecteerd of de slang zichzelf raakt.
        print("□□Game over! De slang raakte zichzelf.")
        exit()

pgzrun.go()
```

?? Uitleg

- `snake = [(x, y), ...]`: lijst van alle slangsegmenten
- `insert(0, new_head)`: voegt een nieuw blokje toe aan de voorkant
- `pop()`: verwijdert het laatste stukje (staart)
- `if new_head in snake[1:]`: controle of slang zichzelf raakt

```
Voorbeeld bij beweging:

Stap 1: slang = [(4,2), (3,2), (2,2)]
Stap 2: kop wordt (5,2) → slang = [(5,2), (4,2), (3,2)]
Stap 3: geen appel → staart eraf → slang = [(5,2), (4,2)]
```

□□ Waarom werkt dit zo?

Door altijd eerst een kop toe te voegen, beweegt de slang vooruit. Als er geen appel is geraakt, halen we de staart weg zodat de lengte gelijk blijft. Wordt er wél een appel geraakt, dan blijft de slang langer: we doen dan geen `pop()`. Dit is hoe de slang groeit!

?? Opdracht

- Controleer of de slang groeit als hij een appel eet
- Controleer of hij even lang blijft als er geen appel wordt geraakt

? Inleveren

1. Beantwoord de vraag: **Wat gebeurt er als je `snake.pop()` vergeet?**

(Lever dit in via het tekstvak of als bestand.)

8 De slang sneller maken met vertraging

In deze les leer je hoe je de slang sneller kunt laten groeien door de `step` te vergroten, zonder dat het spel te snel en onspeelbaar wordt.

? Wat is het probleem?

Je denkt misschien: "Ik wil dat de slang sneller beweegt, dus ik maak `step` groter." Maar als je `step = 20` doet, wordt het spel ineens **veel te snel**.

Dat komt omdat `update()` standaard 60 keer per seconde wordt uitgevoerd. Dus je slang maakt dan 60 sprongen van 20 pixels per seconde: dat is 1200 pixels!

? Fout idee

```
step = 20  # grotere stap
# maar het spel gaat nu te snel!
```

? Goede oplossing: beweging vertragen

We laten de slang **niet elke update** bewegen, maar bijvoorbeeld 1x per 10 frames. Zo kun je `step` groter maken, zonder dat het spel onbestuurbaar wordt.

? Code

```
import pgzrun
import random

WIDTH = 600
HEIGHT = 400
```

```
tile_size = 20
step = 20
vertraging = 10  # aantal frames wachten
frames = 0

snake = [(100, 100)]
richting = "right"

apple_x = random.randint(0, (WIDTH - tile_size) // tile_size) * tile_size
apple_y = random.randint(0, (HEIGHT - tile_size) // tile_size) * tile_size

def draw():
    screen.clear()
    for segment in snake:
        screen.draw.filled_rect(Rect(segment, (tile_size, tile_size)), "green")
    screen.draw.filled_circle((apple_x + tile_size // 2, apple_y + tile_size // 2), tile_size
// 2, "yellow")

def on_key_down(key):
    global richting
    if key == keys.LEFT and richting != "right":
        richting = "left"
    elif key == keys.RIGHT and richting != "left":
        richting = "right"
    elif key == keys.UP and richting != "down":
        richting = "up"
    elif key == keys.DOWN and richting != "up":
        richting = "down"

def update():
    global frames
    frames += 1
    if frames < vertraging:
        return
    frames = 0
    beweeg_slang()

def beweeg_slang():
    global apple_x, apple_y, richting
```

```
    head_x, head_y = snake[0]
    if richting == "left":
        head_x -= step
    elif richting == "right":
        head_x += step
    elif richting == "up":
        head_y -= step
    elif richting == "down":
        head_y += step

    new_head = (head_x, head_y)
    snake.insert(0, new_head)

    head_rect = Rect(new_head, (tile_size, tile_size))
    apple_rect = Rect((apple_x, apple_y), (tile_size, tile_size))

    if head_rect.colliderect(apple_rect):
        print("\U0001F34F Appel geraakt!")
        apple_x = random.randint(0, (WIDTH - tile_size) // tile_size) * tile_size
        apple_y = random.randint(0, (HEIGHT - tile_size) // tile_size) * tile_size
    else:
        snake.pop()

    if new_head in snake[1:]:
        print("\u274C Game over! De slang raakte zichzelf.")
        exit()

pgzrun.go()
```

? Waarom werkt dit?

- `frames += 1`: telt hoe vaak `update()` al is uitgevoerd
- Pas als `frames >= vertraging`, beweegt de slang echt
- `step` bepaalt nu de grootte van de stap, niet de snelheid van het spel

? Opdracht

- Stel `step = 20` in en test of de slang nu sneller groeit
- Experimenteer met `vertraging = 5` of `vertraging = 15`
- Wat is een fijne balans tussen stapgrootte en snelheid?
- Test het spel; wat gebeurt er precies als de slang tegen zijn eigen staart botst?

⬜⬜ Extra uitdaging

Laat de slang naarmate hij langer wordt automatisch steeds iets sneller bewegen (tip: verlaag `vertraging` bij elke appel).

? Inleveren

1. Leg uitwat er gebeurt als de slang tegen zijn staart aankomt.
2. Probeer te vertellen wat je zou willen veranderen; wat moet er gebeuren als de slang tegen zijn slang aan botst?

(Lever dit in via het tekstvak of als bestand.)

9 Score bijhouden en Game Over tonen

In deze les voegen we een score toe aan het spel. Elke keer als je een appel eet, krijg je 1 punt. En als de slang zichzelf raakt, laten we **"Game Over"** op het scherm zien en stopt het spel.

? Wat gaan we doen?

- Een variabele `score` bijhouden
- De score op het scherm tonen met `screen.draw.text()`
- Bij een botsing met jezelf: `game_over = True`
- Het spel stoppen en "Game Over" tonen

? Startercode

```
import pgzrun
import random

WIDTH = 600
```

```
HEIGHT = 400
tile_size = 20
step = 20
vertraging = 10
frames = 0

snake = [(100, 100)]
richting = "right"
score = 0

apple_x = random.randint(0, (WIDTH - tile_size) // tile_size) * tile_size
apple_y = random.randint(0, (HEIGHT - tile_size) // tile_size) * tile_size

game_over = False

def draw():
    screen.clear()
    if game_over:
        screen.draw.text("GAME OVER", center=(WIDTH // 2, HEIGHT // 2), fontsize=60,
color="red")
        screen.draw.text(f"Score: {score}", center=(WIDTH // 2, HEIGHT // 2 + 50),
fontsize=40, color="white")
    else:
        for segment in snake:
            screen.draw.filled_rect(Rect(segment, (tile_size, tile_size)), "green")
        screen.draw.filled_circle((apple_x + tile_size // 2, apple_y + tile_size // 2),
tile_size // 2, "yellow")
        screen.draw.text(f"Score: {score}", topleft=(10, 10), fontsize=30, color="white")

def on_key_down(key):
    global richting
    if key == keys.LEFT and richting != "right":
        richting = "left"
    elif key == keys.RIGHT and richting != "left":
        richting = "right"
    elif key == keys.UP and richting != "down":
        richting = "up"
    elif key == keys.DOWN and richting != "up":
        richting = "down"
```

```
def update():
    global frames
    if game_over:
        return
    frames += 1
    if frames < vertraging:
        return
    frames = 0
    beweeg_slang()

def beweeg_slang():
    global apple_x, apple_y, score, game_over, richting

    head_x, head_y = snake[0]
    if richting == "left":
        head_x -= step
    elif richting == "right":
        head_x += step
    elif richting == "up":
        head_y -= step
    elif richting == "down":
        head_y += step

    new_head = (head_x, head_y)

    if new_head in snake[1:]:
        print("□ Game over! De slang raakte zichzelf.")
        game_over = True
        return

    snake.insert(0, new_head)

    head_rect = Rect(new_head, (tile_size, tile_size))
    apple_rect = Rect((apple_x, apple_y), (tile_size, tile_size))

    if head_rect.colliderect(apple_rect):
        print("\U0001F34F Appel geraakt!")
        score += 1
        apple_x = random.randint(0, (WIDTH - tile_size) // tile_size) * tile_size
        apple_y = random.randint(0, (HEIGHT - tile_size) // tile_size) * tile_size
```

```
    else:
        snake.pop()


pgzrun.go()
```

? Waarom werkt dit?

- We tekenen de score linksboven tijdens het spel
- We zetten `game_over = True` bij een zelfbotsing
- Als `game_over` waar is, stoppen we de beweging en tonen een ander scherm

? Opdracht

- Laat de slang een paar appels eten en kijk of de score klopt
- Laat de slang zichzelf raken en controleer of "Game Over" verschijnt

□□ Extra uitdaging

- Laat de score ook in de console printen bij elke appel
- Laat het spel automatisch herstarten na een paar seconden (of met een toets)

? Inleveren

1. Beantwoord: **Hoe weet de code dat het "Game Over" is en wat gebeurt er dan precies?**
   Leg stap-voor-stap uit wat er gebeurt. Kijk goed naar de code!

(Lever dit in via het tekstvak of als bestand.)

10 Snake – Eindopdracht

?? Houd de slang binnen het scherm

Op dit moment kan de slang zomaar van het scherm verdwijnen als je te ver naar links, rechts, boven of onder beweegt. Bedenk zelf hoe je dat kunt oplossen!

Tip: je kunt controleren of de kop van de slang buiten het scherm terechtkomt. Bijvoorbeeld:

```
if head_x < 0 or head_x >= WIDTH or head_y < 0 or head_y >= HEIGHT:
    game_over = True
```

Op deze manier ben je 'af' als je het scherm raakt. Je zou ook van richting kunnen veranderen, maar dit is wat lastiger om uit te voeren.

?? Opdracht

Zorg ervoor dat de slang niet buiten het beeld kan verdwijnen. Je kunt het natuurlijk zo maken dat als je de rand raakt, het spel klaar is ("Game over").

Een ander optie is om van richting te veranderen. Dat is lastiger, weet je waarom? Hoe zou je het op een goede manier kunnen uitvoeren?

? Inleveren

1. Leg uit waarom het lastig is om de slang van richting te laten veranderen als die de rand van het scherm raakt. Je hoeft het niet te coderen, maar leg uit hoe je de slang precies van richting zou kunnen laten veranderen als die de rand raakt. En werkt dit ook in de hoeken?

   *Probeer te omschrijven hoe het zou kunnen werken. maak als het ware een **AI prompt** (een AI opdracht) om de code op een bepaalde manier aan te passen. Beschrijf **precies** wat er moet gebeuren!*

□□ Extra uitdaging

Als je goed hebt beschreven wat er precies moet gebeuren en je hebt ook rekening gehouden met de hoeken dan kun je misschien AI vragen jouw te helpen om dit echt te bouwen? Probeer maar ! Als het lukt, wordt het spel leuker om te spelen!

--

Introductie AI

1, wat is AI?

datasource

?Introductie

In deze les leren we wat AI is en we gaan kijken naar het verschil van programmeren met en zonder AI.

We kijken naar de kracht van AI maar ook naar de tekortkomingen.

? Leerdoelen

Aan het einde van deze les:

- Kun je uitleggen wat Artificial Intelligence (AI) is.
- Kun je voorbeelden geven van toepassingen van AI in het dagelijks leven en in softwareontwikkeling.
- Kun je het verschil benoemen tussen klassieke ('gewone') code en code gemaalkt door AI.
- Kun je minimaal drie voordelen van het gebruik van AI benoemen.
- Kun je minimaal drie nadelen of risico’s van AI toelichten.
- Kun je uitleggen hoe je AI slim en verantwoord kunt inzetten bij schoolopdrachten of programmeertaken.
- Heb je een eerste indruk gekregen van wat prompt engineering is en waarom het belangrijk is bij het gebruiken van AI.

Bekijk deze video:

https://www.youtube.com/watch?v=QJE_ycgR8E8

https://www.youtube.com/embed/QJE_ycgR8E8

??Opdracht 1

Vat in één tot drie zinnen samen wat de kernboodschap van dit filmpje is.

?Inleveren

1. Maak de samenvatting in van het filmpje in een text document en vul deze in.
   Lever een **TXT** document in.

2, AI toepassingen

AI is veel meer dan alleen ChatGPT. In deze video wordt uitgelegd waarvoor AI kan worden gebruikt.

Bekijk deze video:

https://www.youtube.com/watch?v=stw2upLHCuI

https://www.youtube.com/embed/stw2upLHCuI

?Theorie

AI-toepassingen per taaktype

Hieronder staan concrete voorbeelden van hoe kunstmatige intelligentie (AI) wordt toegepast in verschillende soorten taken

✦ 1. Classificatie

- **Wat is het?**: Het toewijzen van gegevens aan een bepaalde categorie.
- **Voorbeeld**: Een e-mailsysteem dat automatisch bepaalt of een e-mail *spam* is of *geen spam*, op basis van de inhoud, afzender en gebruikte woorden.

✦ 2. Associatie

- **Wat is het?**: Het ontdekken van patronen of combinaties van items die vaak samen voorkomen.

- **Voorbeeld**: Een webshop gebruikt AI om te ontdekken dat klanten die een *laptop* kopen ook vaak een *laptophoes* kopen. Op basis daarvan worden aanbevelingen gedaan: "Andere klanten kochten ook...".

✦ **3. Optimalisatie**

- **Wat is het?**: Het vinden van de beste oplossing uit veel mogelijkheden, vaak onder bepaalde voorwaarden.
- **Voorbeeld**: Een AI-systeem voor routeplanning bepaalt de *snelste bezorgroutes* voor een pakketdienst, rekening houdend met afstand, verkeer en bezorgtijd.

✦ **4. Voorspelling**

- **Wat is het?**: Het voorspellen van toekomstige waarden of gebeurtenissen op basis van eerdere gegevens.
- **Voorbeeld**: Een bakker gebruikt AI om op basis van eerdere verkoopdata te voorspellen hoeveel *brood* er de komende week nodig is.

✦ **5. Creatie**

- **Wat is het?**: Het genereren van nieuwe inhoud of ideeën met behulp van AI.
- **Voorbeeld**: Een AI-systeem zoals ChatGPT of DALL·E kan een *gedicht schrijven* of een *afbeelding maken* op basis van een beschrijving, bijvoorbeeld: "Een robot die schildert in een zonnebloemenveld".

??Opdracht

Bepaal van elk van de voorbeelden bij welk type AI-toep[assing (creatie, assiociatie, optimalisatie, voorpellen, creatie) dit hoort.

1. Netflix geeft je aanbevelingen voor films op basis van wat je eerder hebt gekeken.
2. Een game bepaalt of je gedrag verdacht is en je mogelijk aan het valsspelen bent.
3. Een routeplanner voor je fietsrit kiest de route met de minste verkeerslichten.
4. TikTok voorspelt welke video je het langst gaat kijken en laat die eerder zien.
5. Een AI-programma maakt een unieke profielfoto in cartoonstijl van jou.

?Inleveren

1. Neem de punten over in een text document en vul deze in. Lever een **TXT** document in.

3, AI en 'gewone' computer code.

Wat is het verschil tussen 'gewone' code en AI-code?

Gewone code (klassieke algoritmes) is gebaseerd op vaste instructies: als je A invoert, gebeurt altijd B. Dit maakt het **voorspelbaar** en **betrouwbaar**. Denk aan een rekenmachine of een robotarm in een fabriek die elke minuut exact dezelfde beweging maakt. De computer voert precies uit wat je hebt geprogrammeerd.

AI-code werkt anders. Die is **getraind op heel veel voorbeelden** (zoals tekst, beelden of data) en leert daarvan zelf **patronen te herkennen**. Dat lijkt een beetje op hoe onze hersenen leren. AI is vaak **minder voorspelbaar**, omdat het zelf beslissingen neemt op basis van wat het geleerd heeft. Hierdoor kan het ook **fouten maken**, zeker als het iets nog niet eerder gezien heeft. ChatGPT is bijvoorbeeld een AI die voorspelt welk woord het beste past, op basis van miljarden voorbeelden.

??Opdracht

Opdracht: AI-code of klassieke code? Lees de eigenschappen hieronder. Bepaal of het hoort bij klassieke code of bij AI-code. Zet er een kruisje bij:

Eigenschap	Klassieke code	AI-code	Geen van beiden
Voert altijd precies dezelfde handeling uit			
Kan leren van voorbeelden			
Maakt soms fouten bij onbekende situaties			
Is goed in rekenen en logica			
Kan patronen herkennen			
Is 100% voorspelbaar			
Kan nieuwe dingen maken (zoals een tekening)			
Kan adviseren of je in Bitcoin moet stappen of moet verkopen			
Zal altijd goed advies geven voor de aankoop/verkoop van Bitcoin			
Kan jouw foto veranderen en jou in een hele vreemde situatie zetten			

?Inleveren

1. Neem de tabel over in een Word document en vul deze in. Lever een **PDF** document in.

4, voordelen van AI

?Voordelen van AI

AI heeft een aantal sterke kanten die het nuttig maken in allerlei toepassingen:

Patroonherkenning (Pattern Recognition)

AI kan grote hoeveelheden data bekijken en **herhalende structuren of afwijkingen** vinden.

Beeldherkenning (Computer Vision)

AI kan **beelden of video's analyseren** en daar objecten, mensen of afwijkingen in herkennen.

Voorspellende Analyse (Predictive Analytics)

AI gebruikt **historische gegevens** om te **voorspellen wat er waarschijnlijk zal gebeuren**.

Data-analyse en Forecasting

AI verwerkt en vergelijkt **grote hoeveelheden gegevens over tijd** om een **verwachting voor de toekomst** te berekenen.

Aanbevelingssystemen (Recommendation Systems)

AI leert van jouw **voorkeuren en gedrag** om nieuwe suggesties te doen die bij je passen.
AI vergelijkt **gedrag en voorkeuren** van verschillende gebruikers om te bepalen **wat iemand waarschijnlijk leuk vindt**.

??Opdracht

Hieronder zie je een aantal situaties. Geef per situatie aan of AI hier een voordeel zou kunnen bieden, en leg uit waarom aan de hand van de hierboven genoemde voordelen.

Dus je bedenkt bij welke situatie welk voordeel zou horen.

1. Een docent moet elke week 200 toetsresultaten controleren op fouten.
2. Een leerling zoekt elke dag naar nieuwe muziek die past bij zijn smaak.
3. Een ziekenhuis wil sneller afwijkingen op longfoto's opsporen.
4. Een bedrijf wil weten welke producten waarschijnlijk snel uitverkocht raken.
5. Een bakker wil weten hoeveel broden hij volgende week moet bakken.

?Inleveren

1. Neem de punten over in een text document en vul deze in. Lever een **TXT** document in.

5, nadelen van AI

?Nadelen van AI

Hoewel AI veel voordelen heeft, zijn er ook belangrijke nadelen en aandachtspunten:

- **AI kan fouten maken** als het situaties tegenkomt die het niet kent of niet goed begrijpt.
- **AI is afhankelijk van data** — als de data onvolledig of bevooroordeeld is, kan het systeem verkeerde beslissingen nemen.
- **AI is vaak een ‘black box’** — het is soms moeilijk te begrijpen waarom een AI iets doet of beslist.
- **AI kan banen vervangen**, vooral bij repetitieve taken, wat zorgt voor zorgen over werkgelegenheid.
- **AI heeft geen ethiek of gevoel** — het kan geen morele afwegingen maken zoals mensen dat doen.
- **AI kan misbruikt worden** — bijvoorbeeld voor deepfakes, spam of het beïnvloeden van meningen via sociale media.

??Opdracht: Waar is AI een risico of nadeel?

Hieronder zie je een aantal situaties. Geef per situatie aan of het gebriuk van AI hier een nadeel zou kunnen opleveren, en leg uit waarom aan de hand van de hierboven genoemde voordelen.

Dus je bedenkt bij welke situatie welk nadeel zou horen.

1. Een AI-systeem beoordeelt sollicitaties automatisch en kiest wie wordt uitgenodigd.
2. Een AI-chatbot geeft medisch advies zonder dat een arts meekijkt.
3. Een zelfrijdende auto moet een noodbeslissing nemen in het verkeer.
4. Een bedrijf gebruikt AI om te controleren hoeveel pauze werknemers nemen.
5. Een leerling gebruikt AI om al zijn schoolopdrachten te laten schrijven.

?Inleveren

1. Neem de punten over in een text document en vul deze in. Lever een **TXT** document in.

6, slim gebruik van AI

?Hoe benut je de voordelen en vermijd je de nadelen?

AI is een krachtig hulpmiddel, maar het is belangrijk dat je er **bewust en slim mee omgaat**. Dat betekent: weten wanneer je AI goed kunt inzetten, en ook herkennen wanneer het beter is om zelf na te denken of iets te controleren.

✔ Zo benut je de voordelen van AI:

- Gebruik AI als **assistent**, niet als vervanger van je eigen denkwerk.
- Laat AI je **helpen bij brainstormen**, schrijven of samenvatten — maar **controleer altijd de output**.
- Gebruik AI om **saaie of repetitieve taken te versnellen**, zoals opmaak of vertalingen.
- Combineer AI-output met je **eigen kennis en creativiteit**, zodat het persoonlijk blijft.

✘ Zo vermijd je de nadelen van AI:

- **Geloof niet alles wat AI zegt** — **controleer** feiten en cijfers.
- **Gebruik AI niet voor belangrijke beslissingen** zonder menselijk toezicht.
- **Denk na over privacy en veiligheid** — deel geen persoonlijke gegevens.
- **Gebruik AI niet om te spieken** of werk van anderen als je eigen werk in te leveren
- **Weet wanneer je AI niet moet gebruiken**: Soms is menselijk oordeel belangrijker, bijvoorbeeld bij gevoelige onderwerpen of ethische keuzes.

??Opdracht: Slim of niet slim gebruik van AI?

Lees de onderstaande situaties. Geef per situatie aan of dit **slim gebruik van AI** is of juist **onverstandig**, en leg uit waarom.

1. Een leerling vraagt ChatGPT om 5 ideeën voor een spreekbeurt en kiest daarna zelf het beste idee.
2. Iemand plakt een volledige schoolopdracht in ChatGPT en levert het antwoord in zonder iets aan te passen.
3. Een student laat AI een samenvatting maken van een moeilijke tekst en controleert die daarna met de originele tekst erbij.
4. Iemand vraagt aan een AI wat de beste medicijnen zijn voor zijn klachten, zonder met een arts te praten.
5. Een leerling gebruikt AI om zijn code te verbeteren, maar probeert eerst zelf te begrijpen wat het doet.

?Inleveren

1. Neem de punten over in een text document en vul deze in. Lever een **TXT** document in.

7, Prompt engineering

?Wat is Prompt Engineering?

Prompt engineering is het slim en bewust formuleren van opdrachten of vragen (prompts) voor een AI-systeem zoals ChatGPT, zodat je een zo goed mogelijk en bruikbaar antwoord krijgt.

Als jij aan een mede student vraagt of hij je met de vorige opdracht kan helpen, dan weet hij waarschijnlijk waar je het over hebt. Een AI zoals ChatGPT weet dat niet. Omdat ChatGPT de context niet weet.

> *AI weet niet wat jij denkt!*

De context is alles om een vraag heen.

Voorbeeld

```
Maak een quiz.
```

Deze opdracht is **veel te vaag**. De AI weet niet:

- Over welk onderwerp je quiz moet gaan

- Voor welke doelgroep (leeftijd, niveau)
- Hoeveel vragen je wilt
- Of het meerkeuzevragen of open vragen moeten zijn
- Hoe gedetailleerd het antwoord moet zijn

?Voorbeeld met goede prompt engineering:

```
Maak een quiz van 5 meerkeuzevragen voor leerlingen van 14 jaar
over het onderwerp ‘kunstmatige intelligentie’.

Geef bij elke vraag vier antwoordmogelijkheden en vermeld welk antwoord correct is
```

Deze prompt is **beter** omdat hij:

- context geeft (leerlingen van 14 jaar, onderwerp)
- duidelijk beschrijft het formaat (5 meerkeuzevragen)
- details bevat (vier antwoordkeuzes, juiste antwoord)

We gaan hier in de module prompt engineering verder mee werken, maar we gaan nu vast een oefening doen.

??Opdracht, maak een prompt

(deze opdracht is vervallen ivm met de volgorde van onderwerpen in het 1ste jaar)

Maak één prompt waarin je je AI vraagt een PHP programma dat zoveel mogelijk lijkt op:

image.png

Je hoeft geen database te maken, je hoeft alleen front-end code te maken.

Maak dit met één prompt en wees zo compleet mogelijk.

??Alternatieve opdracht, maak een prompt

Maak de game breakout in Pyton:

- Je start met drie levens
- Plaatjes zoals balletje kan je ook door AI laten maken (via Python Script).

Laat het spel er zoveel mogelijk als volgt uit zien.

image.png

image.png

image.png

? Prompting strategiën

One shot prompt

Bij **one-shot prompting** probeer je in één keer een **zo volledig mogelijke prompt** te schrijven.
Je beschrijft alle specificaties, wensen en details tegelijk.

□□ **Doel:** met één opdracht het hele resultaat (bijvoorbeeld een spel of programma) laten maken.
□□ Daarna gebruik je alleen nog een paar kleine vervolgprompts om details aan te passen.

Iterative prompting

Bij **iterative prompting** werk je **stapsgewijs**.
Je bouwt het project op in **kleine, logische stappen**.

□□ **Doel:** controle houden over het proces en beter begrijpen wat er gebeurt.
□□ Je moet zelf goed bedenken welke stappen je kunt onderscheiden en in welke volgorde je ze uitvoert —
net als bij het bouwen van een huis begin je niet met het dak of de tuin.

Advies

Omdat AI het concept van een *break-out game* al kent, is een **one-shot prompt** goed mogelijk.
Je hoeft dus niet helemaal uit te leggen hoe het spel werkt — de AI vult zelf details aan of “verzint” onderdelen die jij niet benoemt.

□□ **Tip:** probeer je prompt een paar keer te herschrijven en kijk hoe kleine aanpassingen het resultaat veranderen.

□□ **Tip:** vraag AI of jouw prompt voldoende informatie bevat en of alles specifiek genoeg beschreven is. Vraag aan AI wat je nog kan doen om de prompt duidelijker te maken.

Uitdaging

Als je tijd over hebt, is het leuk om een **eigen versie** van het spel te maken.
Bedenk zelf hoe je het spel spannender of leuker kunt maken!

?Inleveren

1. De door jouw gemaakte prompt
2. Het resultaat in code

--

Snake Challenge

Snake Challenge!

datasource

Maak je eigen snake versie!

In deze laatste les ga je zelf aan de slag. Je krijgt een paar uitdagingen waar je zelf oplossingen voor moet bedenken. De code werkt al goed, maar nog niet perfect: je slang kan bijvoorbeeld zomaar uit beeld verdwijnen. En misschien wil je het spel ook leuker of spannender maken?

? Bedenk en bouw je eigen toevoeging

Voeg nu zelf iets toe aan het spel. Kies uit een van de onderstaande ideeën, of verzin er zelf een!

□□ Idee 1: Maak het spel sneller

Verlaag de vertraging een beetje elke keer als je een appel eet. Dan wordt het spel steeds spannender!

```
vertraging = max(1, vertraging - 1)
```

□□ Idee 2: Verander de kleur van de slang bij elke appel

Gebruik bijvoorbeeld `random.choice(["green", "blue", "purple"])` om de kleur van de slang steeds te wisselen.

□ Idee 3: Maak obstakels op het scherm

Voeg een vaste lijst toe met blokken waar de slang niet overheen mag. Als hij ze raakt: game over!

? Extra idee?

Heb je zelf een ander idee? Ga ervoor! Laat je creativiteit zien. Denk aan een "pauze-knop", meerdere appels tegelijk, of een level-systeem.

? Inleveren

1. Maak een korte beschrijving van wat je hebt aangepast (txt bestand)
2. Lever je aangepaste en werkende spel in.

Kennis Check Blok 2

Kennis Check Blok 2.1

datasource

Vallende Stenen

Waarom is het handig om de speler en de vallende steen elk met een eigen variabele voor x en y te beheren?

Zo kun je de positie van elk object onafhankelijk aanpassen en er later berekeningen mee doen, zoals bewegen of botsing detecteren. Als je maar één variabele zou gebruiken, zou je die flexibiliteit verliezen.

Wat is het verschil tussen tekenen op het scherm en het updaten van een positie?

Tekenen op het scherm gebeurt in de `draw()`-functie en bepaalt wat de speler ziet. Updaten van posities gebeurt in de `update()`-functie en bepaalt waar objecten zich bevinden. Zonder update gebeurt er niets; zonder draw zie je niets.

Waarom gebruik je een functie zoals `colliderect()` en schrijf je geen eigen code om te controleren of objecten elkaar raken?

`colliderect()` is getest en betrouwbaar. Zelf botsingsdetectie schrijven is foutgevoelig en complex. Door bestaande functies te gebruiken maak je je code korter, leesbaarder en minder foutgevoelig.

Waarom wordt er gewerkt met `random.randint()` om de x-positie van de steen te kiezen?

Daardoor weet de speler niet waar de volgende steen zal vallen, wat het spel spannender maakt. Zonder toeval zou het spel voorspelbaar en saai worden.

Wat gebeurt er als je geen limiet stelt aan de onderkant van het scherm voor vallende objecten?

Dan vallen stenen eindeloos door en verdwijnen ze uit beeld, wat kan zorgen voor geheugenproblemen of verwarring bij de speler. Je moet ze resetten of verwijderen als ze te ver vallen.

Waarom wordt in het spel vaak gebruik gemaakt van globale variabelen?

Omdat `draw()` en `update()` automatisch worden aangeroepen door Pygame Zero en geen parameters gebruiken, moeten variabelen buiten deze functies beschikbaar zijn. Globale variabelen maken dit mogelijk.

Wat is het verschil tussen een spel dat ‘reageert’ op invoer en een animatie die altijd hetzelfde doet?

Een spel reageert op wat de speler doet, zoals het indrukken van pijltjestoetsen. Bij een animatie loopt alles vanzelf, zonder dat de gebruiker invloed heeft. Interactiviteit is wat van een animatie een spel maakt.

Snake

Waarom bestaat een slang uit een lijst met segmenten in plaats van één positie?

Doordat de slang uit meerdere segmenten bestaat kun je de groei en beweging ervan beter simuleren: bij eten wordt een segment toegevoegd en anders wordt het achterste segment verwijderd. Zo ontstaat een realistische slangbeweging.

Wat is het nut van een timer variabele in `update(dt)`?

De timer bepaalt hoe snel de slang beweegt (bijv. elke 0.15 seconden). Door `dt` op te tellen weet je precies wanneer de slang een stapje moet maken, los van de framerate.

Waarom gebruiken we een `direction_queue` in plaats van alleen de huidige richting?

Een queue maakt het mogelijk om meerdere toetsaanslagen op te slaan vóór de volgende beweging. Zo kan de slang snel achter elkaar van richting veranderen, zonder meteen om te draaien en zichzelf te 'eten'.

Waarom gebruiken we `collide rect()` ?

`collide rect()` is eenvoudiger en betrouwbaarder: het controleert rechthoekige botsingen automatisch. Handmatig rekenen op coördinaten kan fouten veroorzaken.

Wat zou er gebeuren als we de slang laten 'wrappen' naar de andere kant van het scherm bij randbotsing?

Door wrappen speelt het spel vloeiender en blijft de slang ononderbroken bewegen. Zonder wrap zou botsen op de rand een abrupt einde betekenen.

Hoe draagt het willekeurig plaatsen van voedsel bij aan de speelervaring?

Het willekeurig zetten van voedsel voorkomt patronen en houdt het spel uitdagend. De speler moet telkens de slang naar een onverwachte plek sturen.

Wat gebeurt er als de slang op zichzelf botst, en waarom willen we dat checken vóór het toevoegen van een nieuw segment?

Botst de slang op zichzelf, dan eindigt het spel. Door te checken vóór je de kop toevoegt, voorkom je dat de slang in al zijn segmenten terechtkomt — dat voorkomt onverwacht gedrag.

Waarom hebben we een `reset()`-functie die opnieuw alles initialiseert?

Een `reset()`-functie zorgt voor een strak begin na Game Over, met dezelfde beginvoorwaarden (slangpositie, timer, richting). Zo kun je herstarten zonder fouten.

Hoe maak je het spel moeilijker zonder de basiscode te veranderen?

Je kunt de moeilijkheid verhogen door de bewegingstimer in te korten, extra obstakels toe te voegen, of minder ruimte voor voedsel te geven. Zo blijft het uitdagend.

Introductie AI

Wat maakt AI anders dan gewone code?

AI-code is gebaseerd op het leren uit voorbeelden en herkent patronen, terwijl gewone code exact reageert op geprogrammeerde instructies. AI kan dus variabelen voorspellen, terwijl gewone code voorspelbaar blijft :contentReference[oaicite:1]{index=1}.

Waarom onderscheidt men AI-toepassingen in categorieën als classificatie, associatie, enz.?

Zo kun je concrete voorbeelden koppelen aan de manier waarop AI werkt. Elke categorie heeft eigen eigenschappen: classificatie herkent labels, optimalisatie zoekt naar de beste oplossing, creatie gaat over nieuwe inhoud maken, etc. :contentReference[oaicite:2]{index=2}.

Hoe helpt het indelen van AI in typen zoals voorspelling en optimalisatie om je begrip te verdiepen?

Je leert zo dat AI niet één technologie is, maar verschillende werkwijzen heeft afhankelijk van het doel—zoals voorspellen van verkoop of optimaliseren van routes. Dit maakt het makkelijker te kiezen welke techniek geschikt is voor een probleem.
:contentReference[oaicite:3]{index=3}.

Wat betekent het dat AI 'minder voorspelbaar' is dan klassieke code?

Normale code geeft altijd hetzelfde resultaat bij dezelfde input. AI geeft verschillende output, omdat het leert van data en tot een inschatting komt. Hierdoor kan het fouten maken in onbekende situaties :contentReference[oaicite:4]{index=4}.

Waarom is het belangrijk om de voor- en nadelen van AI te begrijpen?

AI heeft veel kracht, zoals patronen herkennen en personaliseren, maar kan ook fouten maken, bevooroordeeld zijn of ondoorzichtig zijn ("black box"). Door deze aspecten te begrijpen kun je AI verantwoordelijk en effectief inzetten. :contentReference[oaicite:5]{index=5}.

Hoe zorgt goede prompt engineering ervoor dat je AI beter helpt begrijpen wat je wilt?

Door context, doel, doelgroep en format duidelijk te geven, krijgt de AI de juiste informatie om relevante en bruikbare output te leveren. Vage prompts leiden vaak tot onsamenhangende antwoorden.

Waarom moet je AI-output altijd controleren, vooral in belangrijke situaties?

AI-systemen kunnen fouten maken, ongepaste of bevooroordeelde antwoorden geven—vooral zonder toezicht. Daarom is menselijke verificatie essentieel, bijvoorbeeld bij medische of juridische adviezen. :contentReference[oaicite:6]{index=6}.

Wat betekent het dat AI een ‘black box’ is, en waarom is dat problematisch?

Je weet vaak niet precies hoe een AI tot een bepaalde beslissing komt. Dit maakt het lastig om die beslissing uit te leggen of te controleren, wat risico's kan veroorzaken in gevoelige toepassingen. :contentReference[oaicite:7]{index=7}.

?? Opdracht

Maak nu de kennis-check.

? Inleveren

Aan het einde van de kennis-check ontvang je een certificaat. Maak een schermafdruk en lever deze in.

Kennis-check Blok 2.2

?? Opdracht

Maak nu de kennis-check.

? Inleveren

Aan het einde van de kennis-check ontvang je een certificaat. Maak een schermafdruk en lever deze in.

Snake lesmatriaal

Deze opdrachten worden gebruikt om in de lessen uiteleg te geven en samen aan de code te werken.

Deel 1

(dit hoort bij 4 van de les)

1. In deze code zit een bug: "de slangenkop" gaat van links naar rechts, maar niet op en en neer. Kijk hoe het werkt, leg uit wat er fout is en pas de code aan.
2. Leg uit hoe de slang van richting veranderd (denk aan kleine stappen wees zo specifiek mogelijk!)

```
import pgzrun

WIDTH = 600
HEIGHT = 400

snake_x = 200
snake_y = 200
tile_size = 20
step = 5
richting = "right"

def draw():
    screen.clear()
    screen.draw.filled_rect(Rect((snake_x, snake_y), (tile_size, tile_size)), "green")

def update():
    global snake_x, snake_y, richting

    if richting == "left":
        snake_x -= step
    elif richting == "right":
        snake_x += step
    elif richting == "up":
        snake_y -= step
```

```
    elif richting == "down":
        snake_y += step

    # Hier controleren we of de slang de rand raakt
    if snake_x < 0:
        richting = "right"
    elif snake_x + tile_size > WIDTH:
        richting = "left"
    elif snake_y < 0:
        richting = "up"
    elif snake_y + tile_size > HEIGHT:
        richting = "down"

def on_key_down(key):
    global richting

    if key == keys.LEFT:
        richting = "left"
    elif key == keys.RIGHT:
        richting = "right"
    elif key == keys.UP:
        richting = "up"
    elif key == keys.DOWN:
        richting = "down"

pgzrun.go()
```

Deel 2

(deze hoort bij de laatste opdracht 10)

We maken een sprongetje in de opdrachten en tekenen nu een slang die wat langer is.

1. Test de code en maak de slang nog een beetje (minimaal 2 blokjes) langer.
2. Oops dezelfde fout als zit weer in de code (stupid programmer!) - pas dit weer aan zodat de slang op ale randen 'bounced'

We 'bouncen' nu op de zijkanten, in *Snake* kan je het ook zo maken dat als je op de rand botst het "Game over" is.
'bouncen' kan niet omdat je dan je eigen staart raakt en dat mag niet in de Snake game.

Question: Stel we willen niet bouncen, en geen Game Over bij het contact van de rand, hoe zou het moeten dan moeten werken?

Beschrijf stap-voor-stap hoe het zou moeten werken in de vorm van een AI prompt.

```
import pgzrun

WIDTH = 600
HEIGHT = 400

tile_size = 20
step = 5
richting = "right"

# Slang met 4 blokjes (array in array: lijst van lijsten)
snake = [[200, 200], [180, 200], [160, 200], [140, 200]]

def draw():
    screen.clear()
    for position in snake:
        x = position[0]
        y = position[1]
        screen.draw.filled_rect(Rect((x, y), (tile_size, tile_size)), "green")

def update():
    global snake, richting

    # huidige kop
    head_x = snake[0][0]
    head_y = snake[0][1]

    if richting == "left":
        head_x -= step
    elif richting == "right":
        head_x += step
    elif richting == "up":
        head_y -= step
    elif richting == "down":
        head_y += step
```

```
    # nieuwe kop aan het begin van de lijst
    new_head = [head_x, head_y]
    snake = [new_head] + snake[:-1]   # schuif de slang door

    # Randcheck
    if head_x < 0:
        richting = "right"
    elif head_x + tile_size > WIDTH:
        richting = "left"
    elif head_y < 0:
        richting = "up"
    elif head_y + tile_size > HEIGHT:
        richting = "down"

def on_key_down(key):
    global richting

    if key == keys.LEFT:
        richting = "left"
    elif key == keys.RIGHT:
        richting = "right"
    elif key == keys.UP:
        richting = "up"
    elif key == keys.DOWN:
        richting = "down"

pgzrun.go()
```