

games maken en ervaren

domein J: keuzethema programmeerparadigma's

domein P: keuzethema user experience

René van der Veen (Augustinuscollege Groningen)

Vincent Velthuis (RuG Groningen)

Ron Smiers (Regiuscollege Schagen)

Adriaan Dekker (Pieter Zandt Scholengemeenschap Kampen)

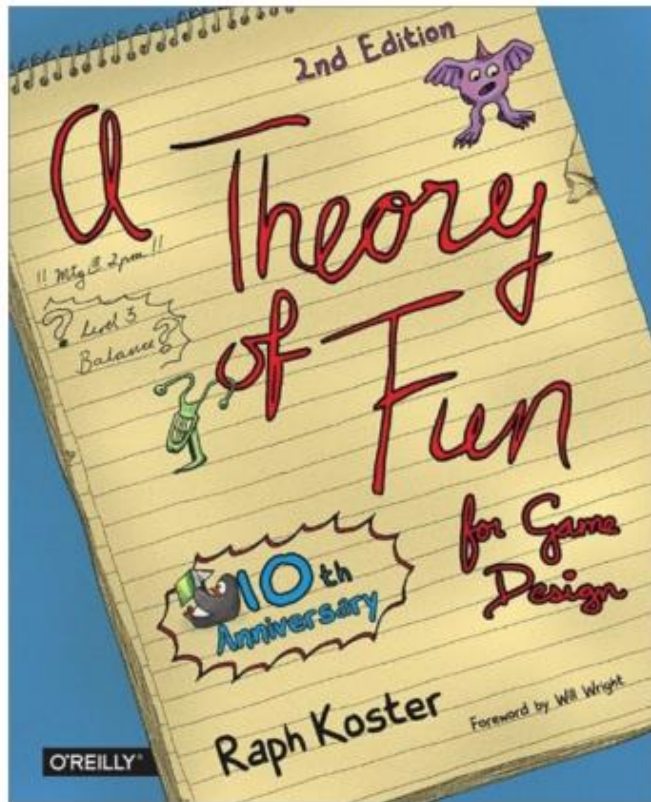
wat gaan we doen in 60 min?

- ☐ games, OO en hun plaats in het examenprogramma
- ☐ onderbouwing van (vak-) didactische keuzes
- ☐ flowcharts en design patterns
- ☐ Javascript en de P5-library
- ☐ opzet van de module
- ☐ hoe gebruik je de module in de klas: keuzes
- ☐ game theorie en een voorbeeld
- ☐ keuzes en dilemma's m.b.t. de stijl van programmeren

Wat is een game?

Theory of Fun for Game Design

Auteur: [Raph Koster](#) | Taal: Engels | ★★★★★ 1 review | [✉ E-mail deze pagina](#)



Auteur: [Raph Koster](#), [Raph Koster](#)
Co-auteur: [Raph Koster](#), [Ralph Koster](#)
Uitgever: [O'Reilly Media, Inc, Usa](#)

A game is a system in which players engage in an abstract challenge, defined by rules, interactivity and feedback, that results in a quantifiable outcome often eliciting an emotional reaction.

- Raph Koster



THE VERGE

*THIS, OF COURSE, IS FORTNITE —
BUT NOT AS YOU KNOW IT*

GAMING

HOW A FORTNITE SQUAD OF SCIENTISTS IS HOPING TO DEFEAT CLIMATE CHANGE

Finally, a good excuse to play video games

By [Robin George Andrews](#) | Oct 10, 2018, 11:52am EDT

Wat is **OO**?

What is Object Oriented Programming?

OOP refers to a type of computer programming (software design) in which programmers define not only the data type of a data structure, but also the types of operations (functions) that can be applied to the data structure.

Hoe past deze module in
het totale aanbod en het
examenprogramma?

domein J:

programmeerparadigma's

J1: alternatief programmeerparadigma De kandidaat kan van minimaal één extra programmeerparadigma de kenmerken beschrijven en kan programma's volgens dat paradigma ontwikkelen en evalueren.

J2: keuze van een programmeerparadigma De kandidaat kan voor een gegeven probleem een afweging maken tussen paradigma's voor het oplossen ervan.

domein P:

user experience

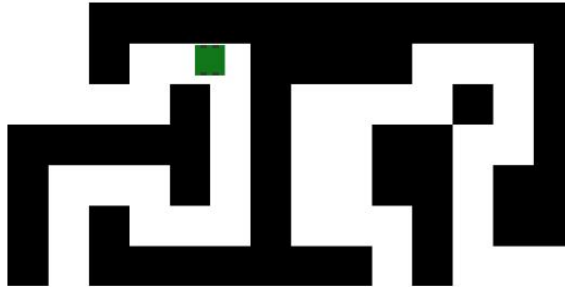
P1: analyse De kandidaat kan de relatie tussen ontwerpkeuzes van een interactief digitaal artefact en de verwachte cognitieve, gedragsmatige en affectieve veranderingen of ervaringen verklaren.

P2: ontwerp De kandidaat kan voor een digitaal artefact de gebruikersinteractie vormgeven, de ontwerpbeslissingen verantwoorden en voor een eenvoudige toepassing implementeren.

domein J
programmeerparadigma's:
functioneel programmeren

domein G
Algoritmiëk, berekenbaarheid & logica:
Algoritmiëk

OO games J + P



domein P
user experience:
sociale robotica

domein P
user experience:
3D gaming met Unity

de opdracht:

Ontwikkel een OO-module
die niet te theoretisch is en
toegankelijk voor havo is.

Hoe maken we **verstandige** keuzes?

3. Leren programmeren

Leren programmeren is moeilijk: het resultaat van programmeervakken of -onderdelen is vaak minder goed dan leerlingen verwachten [20]. Daarbij is de spreiding van de vaardigheden onder de leerlingen vaak groot: naast leerlingen die grote moeite hebben met het vak, maken sommige leerlingen zich het programmeren zeer snel eigen. Al jarenlang proberen onderzoekers de oorzaken van de variabele leeropbrengst te verklaren en te zoeken naar mogelijkheden om het leren programmeren te verbeteren [21]. Dit hoofdstuk beschrijft een aantal belangrijke manieren om tegen het leren programmeren aan te kijken.

Rendement
van
objectgeoriënteerd programmeeronderwijs

Ernst Koldenhof
Johan Jeuring
Sandra Ruth

- Hoe kan verklaard worden dat leerlingen het programmeren na afloop van het onderwijs gemiddeld genomen minder leuk vinden/inschatten dan voor het onderwijs?
- Kan meer aandacht aan het programmeerproces helpen bij het verminderen van *stop*-gedrag bij de leerlingen?
- Waarom ervaren leerlingen informatica als een relatief eenvoudig vak?
- Hoe kunnen we informatica aanbieden zodat zowel de minder goede leerlingen als de goede leerlingen goed worden bediend en uitgedaagd?

Rendement
van
objectgeoriënteerd programmeeronderwijs

Ernst Koldenhof
Johan Jeuring
Sandra Ruth

Andere onderzoekers volgen vooral beginnende programmeurs tijdens het leren programmeren. Perkins et al. [19] observeren dat beginners bij het leren programmeren twee soorten reacties kunnen vertonen t.o.v. de opdrachten en leerstof. De eerste soort reactie is die van iemand die niet meer weet wat hij of zij moet doen, en als reactie daarop maar helemaal niets doet. Deze zogenaamde *stop*-reactie wordt door sommigen gekoppeld aan karaktereigenschappen van de beginnende programmeur. Zo koppelen Murphy et al. [17] dit gedrag aan het zelfbeeld van de beginnende programmeur die in het geval van een *stop*-reactie denkt dat hij/zij opdrachten niet kan omdat hij/zij niet slim genoeg is. De leerling hanteert een beeld waarbij kennis en vaardigheden een vaste eigenschap is (*fixed mindset*). De andere soort reactie is die van een leerling die ondanks het feit dat hij of zij geen duidelijk aanwijzingen heeft wat hij of zij precies moet doen, wel verder gaat. Hierbij volgt of ontwikkelt de leerling een eigen strategie. Een leerling met dit gedrag noemt Perkins een *mover*. Een *mover* maakt veel meer vorderingen dan een *stopper*. Volgens Murphy bezit een *mover* een zelfbeeld waarbij hij of zij meent dat door inspanning de meeste problemen te overwinnen zijn (*growth mindset*). Murphy stelt dat een docent en een leermethode (en met name de opgaven binnen de leermethode) een belangrijk sturend aandeel hebben in het bepalen van dit zelfbeeld. Een leerling kan in verschillende stadia van een opdracht gestimuleerd worden om van een *fixed mindset* naar een *growth mindset* te gaan.

niet

te veel en te zware theorie

maar

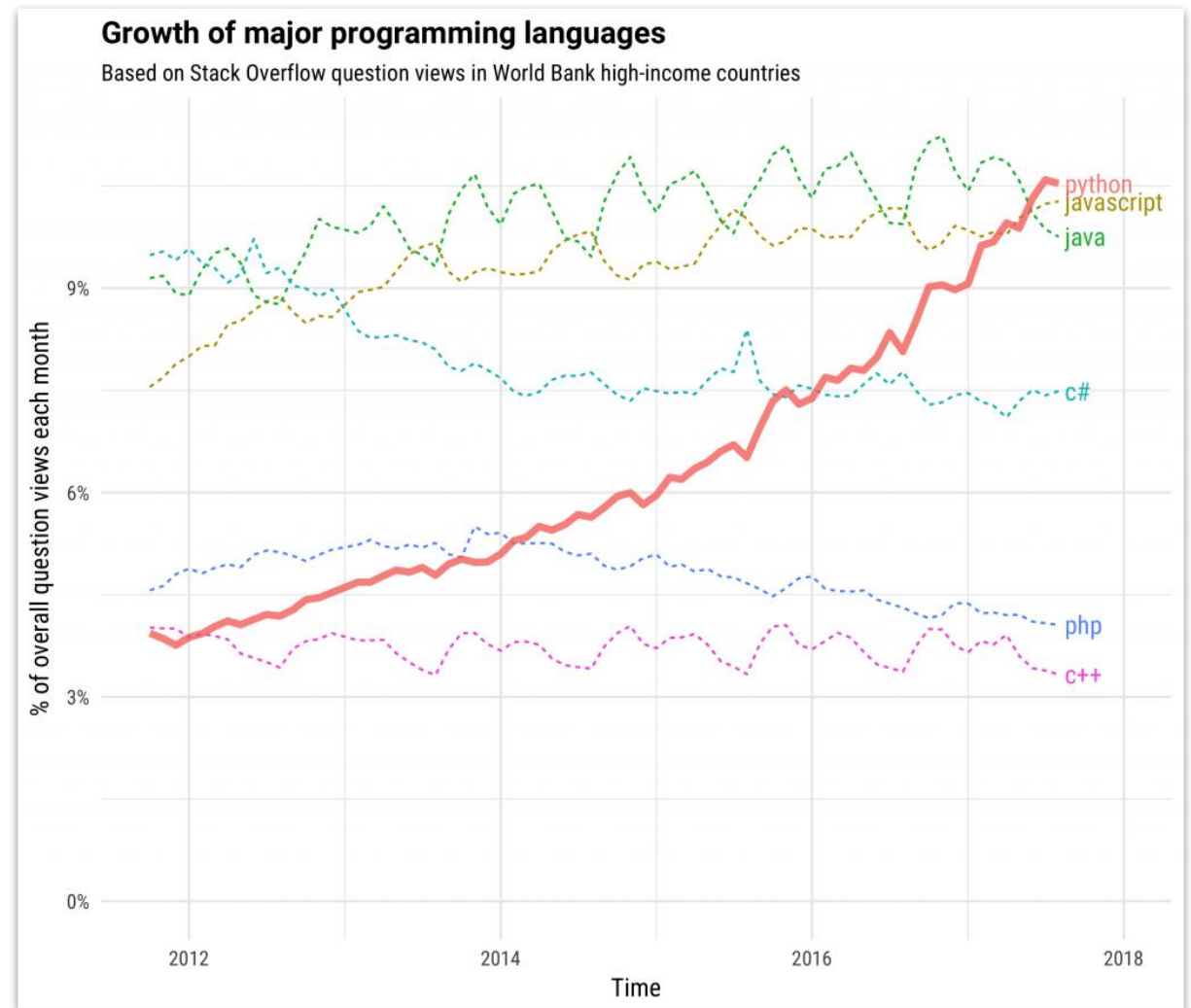
snel zelf aan de slag met resultaat
(het werkt maar ook zelfvertrouwen)

wel implementatie,
niet te veel algoritmes

keuze voor javascript

keuze voor scripttaal Javascript

- Wordt wereldwijd veel gebruikt
- Wordt al heel lang gebruikt
- Heel veel bronnen / ondersteuning
- Heel veel libraries
- Ruime mogelijkheden ter verdieping, ook bij andere (keuze-) domeinen
- Zeer flexibel
- Brede inzetbaarheid
- Werkt overal
- Niet eerst compileren
- Support door code-editors (foutdetectie, code markeren)
- Verschillende tools voor blokprogrammeren (onderbouw) kunnen vertaalslag naar JS maken
- Toegankelijk, laagdrempelig
- Geen apart programma nodig



didactische keuzes:
volgorde en opbouw

Rethinking of Teaching Objects-First

CHENGLIE HU

Department of Computer Science, Carroll College, 100 N East Ave., Waukesha, WI 53186, USA

E-mail: chu@cc.edu

Abstract

The issues surrounding teaching object-orientation to novice programmers from day one are revisited first. An analysis is then presented showing the harmfulness of teaching objects-first. The attention then is given to addressing the problems of the current textbooks. Furthermore, a remark is made in comparison between teaching objects-first and Reformed Calculus from a closed discipline. Finally, a new structure for introductory programming courses is suggested.

6. Conclusion

Teaching objects-first may well be (and in fact, is, in many schools) counter-productive and harmful. On the other hand, given the industry demand, students must experience objects early on. Based on the recommendation of ACM Computing Curricula (2001), a three course sequence for the computer science foundation is necessary. While CS2 (Data Structures) may largely be intact, CS1 should be divided into two courses.

Analysis of Learning Objectives in Object Oriented Programming

Peter Hubwieser

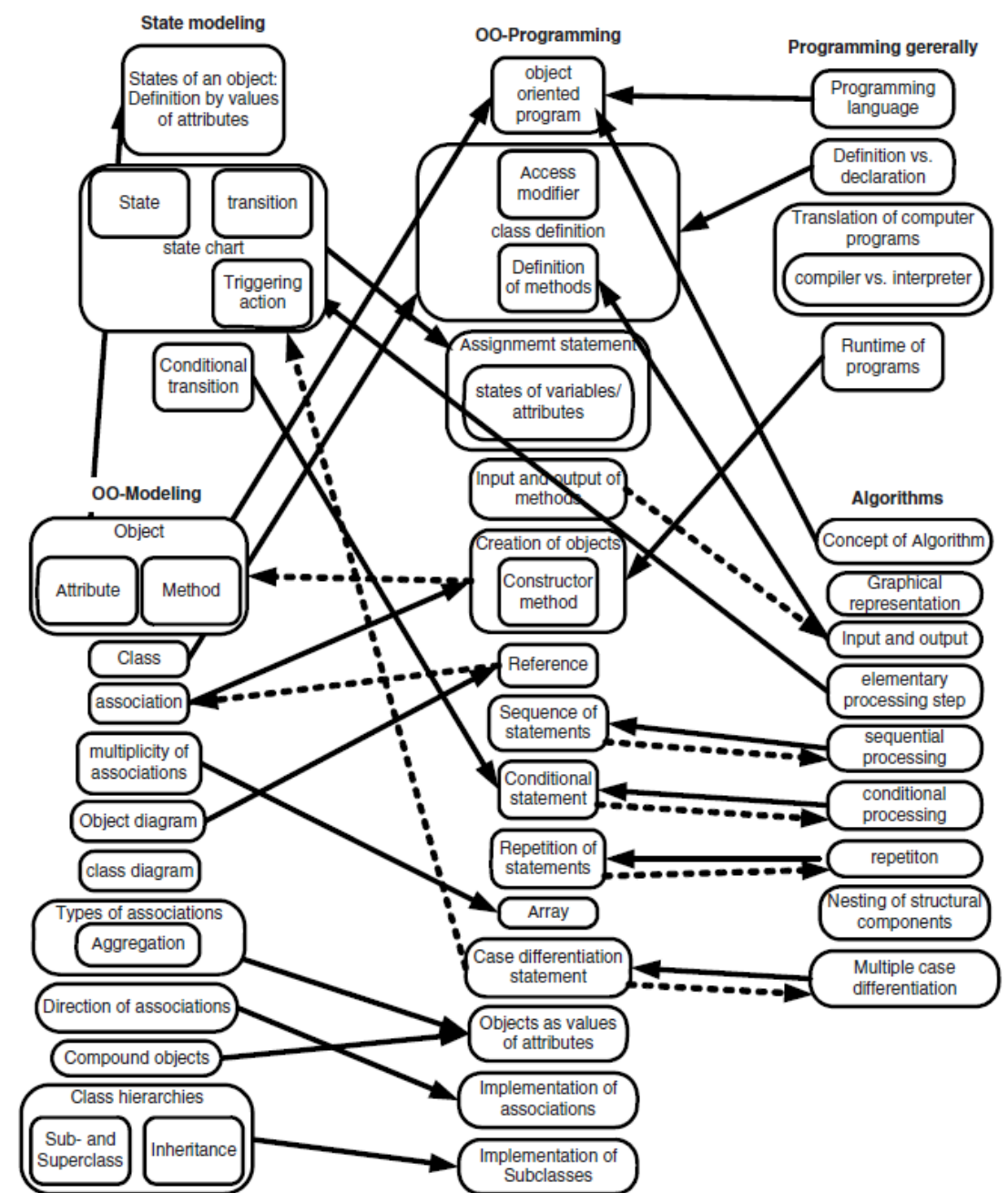


Fig. 1. Prerequisite relations P1 (unbroken arrows) and P2 (dotted arrows) on the learning objectives of OOP

Problem Solving and Algorithmic Development with Flowcharts

Renske Smetsers-Weeda
Radboud University
Nijmegen, The Netherlands
Renske.Smetsters@science.ru.nl

Sjaak Smetsers
Radboud University
Nijmegen, The Netherlands
S.Smetsters@cs.ru.nl

ABSTRACT

Programming, where problem solving and coding come together, is cognitively demanding. Whereas traditional instructional strategies tend to focus on language constructs, the problem solving skills required for programming remain underexposed.

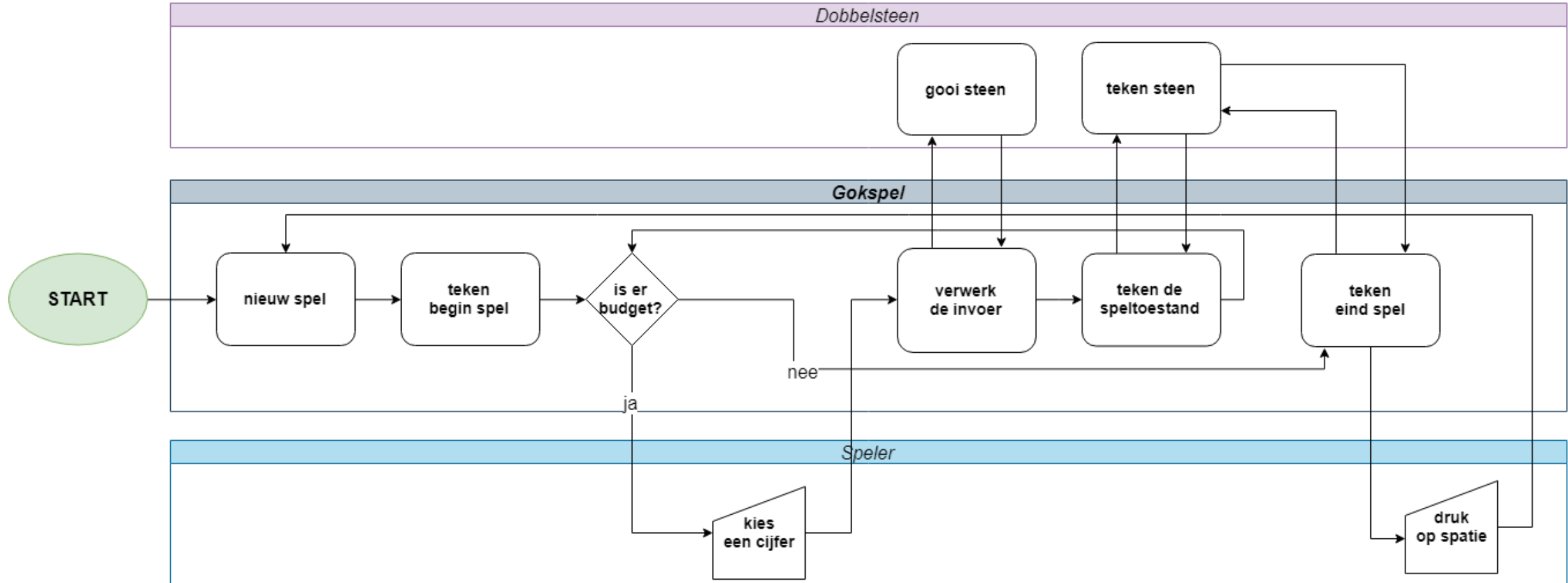
In an explorative small-scale case study we explore a "thinking-first" framework combined with stepwise heuristics, to provide students structure throughout the entire programming process.

Using unplugged activities and high-level flowcharts, students are guided to brainstorm about possible solutions and plan their algorithms before diving into (and getting lost in) coding details. Thereafter, a stepwise approach is followed towards implementation. Flowcharts support novice programmers to keep track of where they are and give guidance to what they need to do next, similar to a road-map.

High-level flowcharts play a key role in this approach to problem solving. They facilitate planning, understanding and decomposing the problem, communicating ideas in an early stage, step-wise implementation and evaluating and reflecting on the solution (and approach) as a whole.

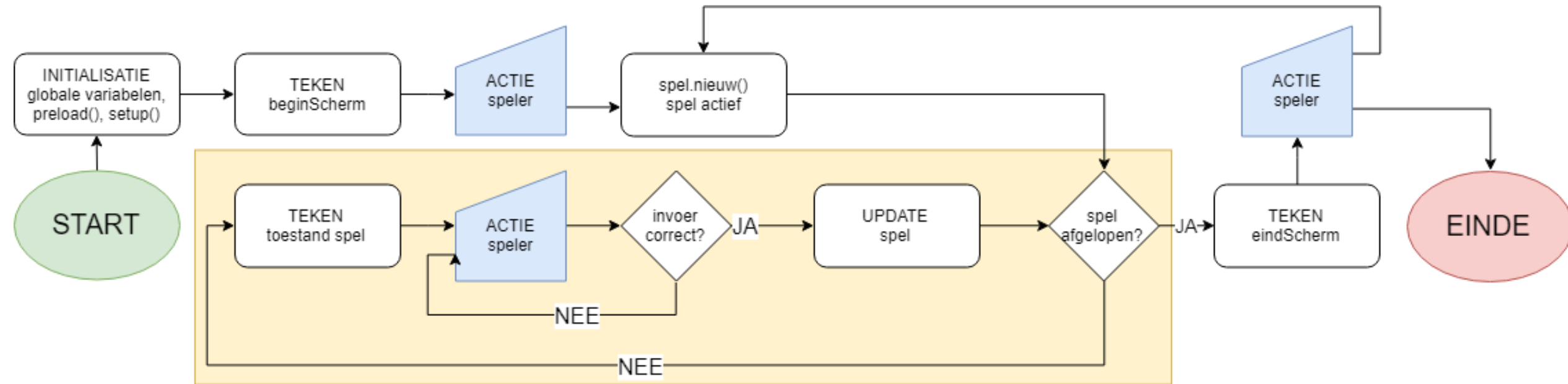
geen eindige automaten,
wel flowcharts

concreet: inclusief de rol van objecten



Design patterns capture solutions that have developed and evolved over time. Hence they aren't the designs people tend to generate initially. They reflect untold redesign and recoding as developers have struggled for greater reuse and flexibility in their software. Design patterns capture these solutions in a succinct and easily applied form. (Gamma et al. 1995, xi.)

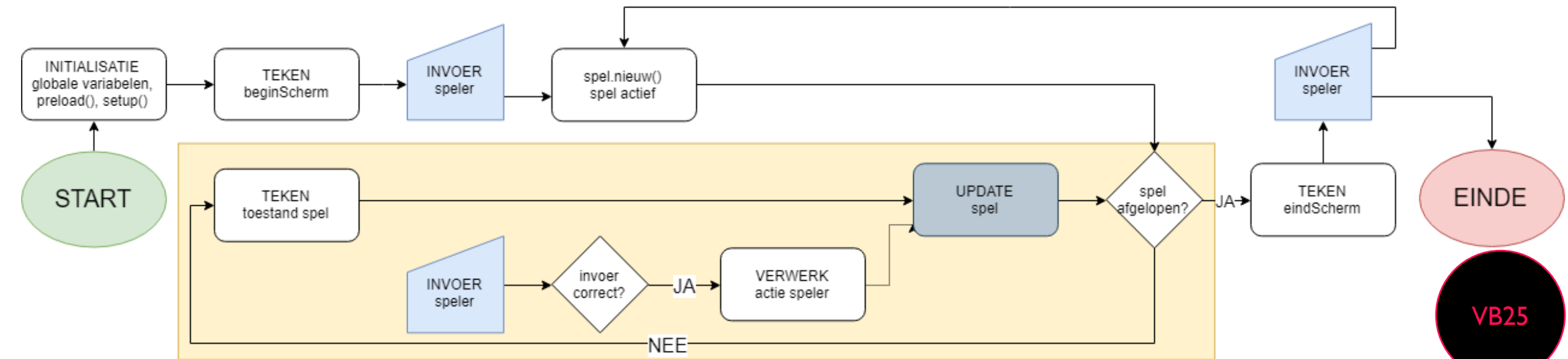
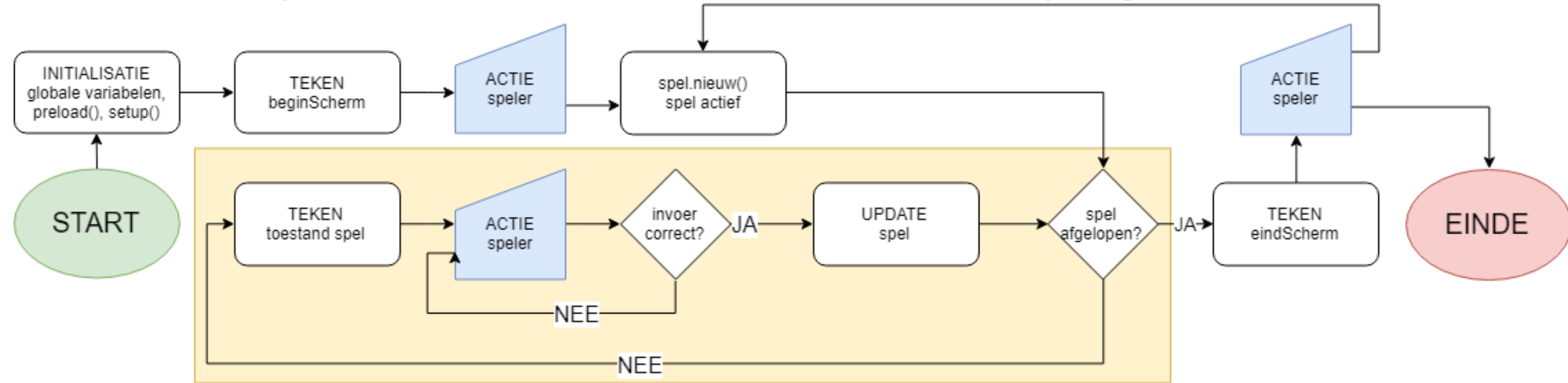
algemenere structuren: design patterns



Kunnen we code & flowchart
parallel laten lopen?

Een flowchart is nog geen technische
implementatie, maar dient als hulpmiddel.

algemenere structuren: design patterns



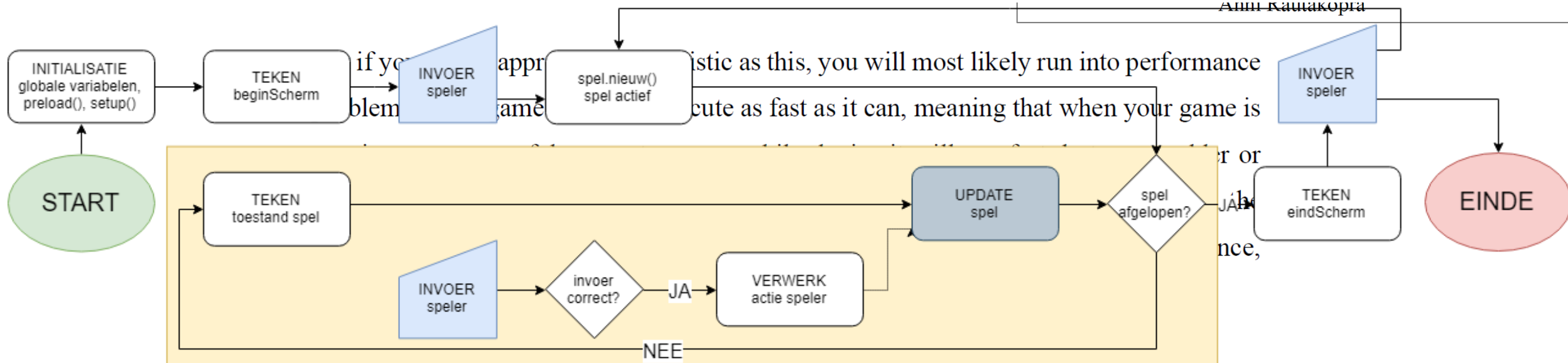
4.1.1 Structure

In its most simplified form a game loop handles user input, advances the game simulation

A game loop has two key pieces: non-blocking user input and adapting to the passage of time. Input is straightforward. The magic is in how you deal with time. There are a near-infinite number of platforms that games can run on, and any single game may run on quite a few. How it accommodates that variation is key. (Nystrom 2009-2014, Game Loop.)

ming

Amir Kaulakopra



4.1.1 Structure

In its most simplified form a game loop handles user input, advances the game simulation one step and draws the result on screen so the player can see what happened. This process is then repeated indefinitely, until the game ends. A very simple example of a game loop might look something like:

```
while(isGameOn) {  
    processInput();  
    update();  
    render();  
}
```

But if you use an approach as simplistic as this, you will most likely run into performance problems. Your game loop will execute as fast as it can, meaning that when your game is running on a powerful computer or a mobile device it will run fast, but on an older or slower device it will run slower. This will create a difference in gameplay, making the player move in different speed depending on the underlying platform used for instance, making network gaming quite unfair.

GAME DESIGN PATTERNS

Utilizing Design Patterns in Game Programming

Anni Rautakopra

keuze voor P5 processing

keuze voor library P5

- (Game-) loop voorgeprogrammeerd
- *Library* ontwikkeld voor en door het onderwijs
- Zeer laagdrempelig, voorgestructureerd
- Snel aantrekkelijk resultaat
- visuele feedback = didactisch sterk
- Wegblijven van *programmeer de tafel van 28*
- Tal van uitbreidingen
- Actieve gemeenschap
- Duidelijke *reference* met veel voorbeelden
- Veel online bronnen en literatuur

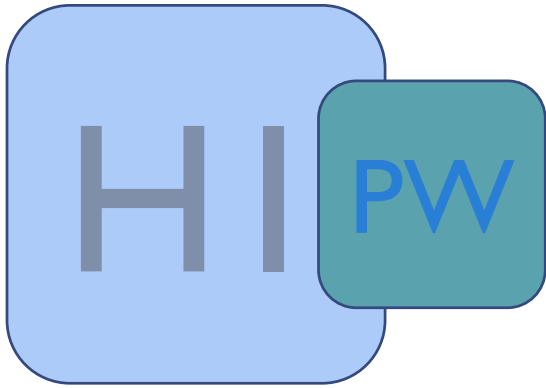
Home	Examples		
Download	Structure	Simulate	Typography
Start	Coordinates	Forces	Letters
Reference	Width/Height	Particle System	Words
Libraries	Setup/Draw	Flocking	3D
Learn	No Loop	Wolfram CA	Geometries
Examples	Loop	Game of Life	Sine Cosine in 3D
Books	Redraw	Multiple Particle	Multiple Lights
Community	Functions	Systems	Materials
	Recursion	Spirograph	Textures
	Create Graphics	L-Systems	Orbit Control
Forum	Form	Spring	Input
GitHub	Points/Lines	Springs	Clock
Twitter	Shape Primitives	Soft Body	Constrain
	Pie Chart	SmokeParticles	Easing
	Regular Polygon	Brownian Motion	Keyboard
	Star	Chain	Mouse 1D
	Triangle Strip	Snowflakes	Mouse 2D
	Bezier	Penrose Tiles	Mouse Press
	3D Primitives	Recursive Tree	Mouse Functions
	Data	The Mandelbrot Set	Mouse Signals
	Variables	Koch Curve	Storing Input
	True/False	Interaction	Advanced Data
	Variable Scope	Tickle	Load Saved JSON
	Numbers	Follow 1	Sound
	Arrays	Follow 2	
		Follow 3	
		Snake game	

P5

kleine demo

opzet lesmateriaal

module games maken en ervaren



- kennismaken met JS en P5
- basisprincipes programmeren
- loop-functie
- (eerste) interactie
- (verdieping) recursie

- **niet** while
- **niet** case

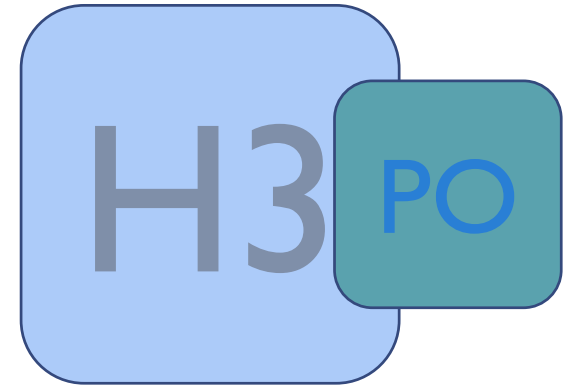
Opfrissen & P5



- standaard objecten (array, afbeelding)
- sprites & spritesheets (verdieping)
- zelf objecten maken
- een object dat antwoordt
- klasse van objecten
- array van objecten (van één klasse)

- **niet** overerving

OO Paradigma



- structuur van een spel: flowcharts
- object spel
- events, acties, gebeurtenissen
- design patterns
- spelregels
- levels
- tijd
- geluid

Structuur & Ervaring

Opdracht 10 simpel tekenprogramma

We gaan de P5-variabelen `mouseX` en `mouseY` gebruiken om zelf een tekening te maken.

54. Open `H1O10.js` in jouw editor. Bekijk het resultaat in de browser. Hoe heeft de programmeur er zonder hoofdrekken voor gezorgd dat de stip precies in het midden staat?
55. Pas de argumenten van de functie `ellipse` aan, zodat de stip jouw muis volgt.

Het is je misschien opgevallen dat `background('lavender');` in dit programma niet in de `setup` maar in de `draw` is geplaatst. Waarom?

56. Verplaats de genoemde regel van de `draw` naar de `setup`. Wat is er veranderd?

Nu de achtergrondkleur in de `setup` wordt ingesteld, wordt deze alleen in het begin gekleurd. Alles wat daarna verandert, blijft in beeld staan.

We kunnen nu tekenen, maar de tekst bovenaan wordt onleesbaar. Gelukkig is er een simpele oplossing:

57. Voeg aan het begin van de `draw` een regel toe die een rechthoek tekent met de (vul-) kleur `wheat`. Gebruik de standaard P5-variabele voor de canvasbreedte en zorg voor een hoogte van 30 pixels zodat bovenaan een balk wordt getekend zoals in figuur 1.17.



FIGUUR 1.17

1.4 draw is een loopfunctie

Alle programma's en voorbeelden die je tot nu toe hebt gezien hebben een `setup` met de regel `noLoop();`. Was het je opgevallen dat deze regel in de laatste twee opgaven was uitgeschakeld met `//`?

In paragraaf 1.1 is genoemd dat in de `setup` de begininstellingen van het programma worden beschreven en dat de `draw` het hoofdprogramma is. In P5 is `draw` wel een bijzonder geval, want de code binnen de `draw` wordt telkens opnieuw uitgevoerd. Een functie die regels code steeds opnieuw herhaalt noemen we een *loopfunctie* (*loop* is Engels voor 'lus' of 'herhalingslus'). In opgave 9 en 10 hebben we voor het eerst het voordeel van zo'n herhalingslus gezien: omdat de `draw` steeds opnieuw wordt uitgevoerd, kan worden gevolgd wat de actuele plaats van de muis is. Met die informatie krijgt de canvastekening steeds een update. Hoe vaak gebeurt dat eigenlijk?

In voorbeeld 5 (waarvan figuur 1.18 een screenshot toont) zie je hoe een cirkel zich van links naar rechts verplaatst. Hierbij gebruikt men:

- o in de `setup`: `frameRate(10);`
Dit zorgt ervoor dat de animatie met 10 frames (beeldjes) per seconde wordt getekend, ofwel: dat de functie `draw` 10x per seconde wordt uitgevoerd.
- o aan het einde van de `draw`: `horizontaal += 2;`
Dit zorgt ervoor dat de variabele `horizontaal` die bepaalt op welke breedte de cirkel wordt getekend elk frame met 2 wordt opgehoogd, zodat de cirkel steeds verder naar rechts wordt getekend.

De regel `horizontaal += 2;` had ook geschreven kunnen worden als `horizontaal = horizontaal+2;` Dat is iets langer, maar voor sommige mensen wel duidelijker. Je moet deze regel lezen als: *de nieuwe waarde van de variabele 'horizontaal' is gelijk aan de oude waarde van de variabele 'horizontaal' plus 2*. Javascript heeft handige, korte notaties voor berekeningen. Wil je één optellen of aftrekken van de waarde van een variabele, dan gebruik je `horizontaal++`; en `horizontaal--`; Verdubbelen? Gebruik `horizontaal *= 2;` Halveren? Gebruik `horizontaal /= 2;` (of `horizontaal *= 0.5;`)

Het is niet altijd nodig om de `draw`-functie eindeloos uit te voeren. Als de code binnen de `draw` maar één keer moet worden uitgevoerd of als je het herhalen wilt stopzetten, dan gebruik je `noLoop();`

✓ Opdracht 11 automatische bewegingen

58. Open `H1O11.js` in jouw editor. Dit is de code van voorbeeld 5. Bekijk het resultaat in de browser.

De blauwe cirkel noemen we cirkel A. Voor de beginpositie van A gebruiken we de variabele `horizontaalA`. We gaan een tweede cirkel B maken. Zie figuur 1.19.

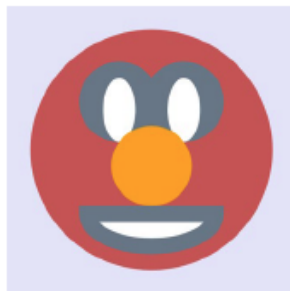


FIGUUR 1.19

59. Declareer voor de beginpositie van B een variabele `horizontaalB` met de waarde 500.
60. Teken cirkel B even groot als A, maar met de kleur `darkred`. Natuurlijk maak je voor de horizontale positie gebruik van `horizontaalB`.
61. Zorg dat de positie van B elke keer dat de `draw` wordt uitgevoerd met 1 wordt verhoogd.
62. Pas het argument van de functie `frameRate` aan naar 50. Wat is het resultaat?
63. Pas de tekst aan zodat er komt te staan:
`positie A = 454 positie B = 667` (Dit is maar één voorbeeld: de waarden veranderen steeds.)

Opdracht 12 maak kennis met JOS

In deze opgave maken we kennis met JOS. JOS is een *game character*: een poppetje dat we in meerdere opgaven tegen zullen komen (figuur 1.20). Zijn naam is een afkorting van *JavaScript Object Sprite*. Een *sprite* is veelgebruikte term voor een tweedimensionaal plaatje of animatie. In hoofdstuk 2 leer je wat een object is.



FIGUUR 1.20

64. Open `H1O12.js` in jouw editor. Bekijk het resultaat. Je ziet een behoorlijk aantal regels die ervoor zorgen dat JOS wordt getekend, maar het is niet nodig om die nu te bestuderen.
65. Voeg de regel `xJOS --;` toe aan het eind van de `draw` en bekijk het resultaat. Wat betekent deze regel?
66. Gebruik `yJOS --;` zodat JOS naar linksboven beweegt.
67. Pas de regel aan naar `yJOS -= 2;` zodat JOS twee keer zo snel omhoog beweegt als dat hij naar links beweegt.
68. Pas regel 17 (`translate`) aan, zodat JOS meebeweegt met jouw muis.

Opdracht 13 P5-functies voor beperken en schalen

In de vorige opdracht was het mogelijk om JOS van het canvas te laten verdwijnen. In veel spellen is het de bedoeling dat je *game character* zichtbaar blijft, ofwel: op het canvas blijft. Voor jou als programmeur betekent dit dat je de beweging van JOS moet inperken.

69. Open `H1O13.js` in jouw editor. Bekijk het resultaat. Zorg dat je de muis heen en weer beweegt!

Als het goed is heb je gemerkt dat je JOS wel kunt bewegen, maar dat je niet meer alle vrijheid hebt. Oorzaak is de regel `xJOS = constrain(mouseX, 100, 450);` (`constrain` is Engels voor beperken). JOS mag bewegen volgens de horizontale muispositie (`mouseX`), maar alleen tussen de waarden 100 en 450.

70. Pas de code aan, zodat JOS links en rechts precies tot de rand van het canvas kan bewegen. HINT: in welke regel kun je zien hoe groot JOS is?
71. Breid de code uit, zodat JOS ook boven en onder precies tot de rand van het canvas kan bewegen.

Als we Jos wat kleiner willen tekenen, lijkt dat een hele klus, omdat hij is opgebouwd met flink wat regels programmeercode. Gelukkig heeft P5 een functie om de omvang van tekeningen te schalen: `scale(1);` Met de waarde 1 wordt alles op normale grootte (100%) getekend, met b.v. `scale(0.5);` op 50%.

72. Teken Jos op 50% van zijn normale grootte. Wat zie je? LET OP: volgt Jos de muis nog wel goed?
73. Verplaats de regel `scale(0.5);` zodat hij meteen na `push();` staat. Probleem opgelost?

1.8 VERDIEPING: recursie ★

In de vorige opdracht heb je het tekenen van een piramide geprogrammeerd. In figuur 1.50 staan de coderegels die we daarvoor hebben gebruikt. De code in figuur 1.50 levert dezelfde piramide op.

```
function draw() {  
  for (var laag=1; laag<=aantalLagen; laag++){  
    tekenRij(laag);  
    translate(0, hoogte);  
  }  
}  
  
function tekenRij(aantalStenen) {  
  inspringen =  
    (aantalLagen - aantalStenen) * 0.5 * breedte;  
  push();  
  translate(inspringen, 0);  
  for (var steen=0; steen<4; steen++) {  
    rect(breedte * steen, 0, breedte, hoogte);  
  }  
  pop();  
}
```

FIGUUR 1.50

Hoewel beide codes hetzelfde eindresultaat opleveren, is de gebruikte programmeertechniek totaal anders.

De variant rechts gebruikt recursie. Recursie is een programmeertechniek die je gebruikt in situaties waarbij een opdracht logisch opgedeeld kan worden in meerdere opdrachten die hetzelfde zijn qua vorm, maar dan eenvoudiger.

Kenmerken voor recursie is dat er een functie is die zichzelf aanroept. De functie `tekenPiramide` in figuur 1.50 bevat zelf weer de coderegel `tekenPiramide`. Het idee daarachter is als volgt:

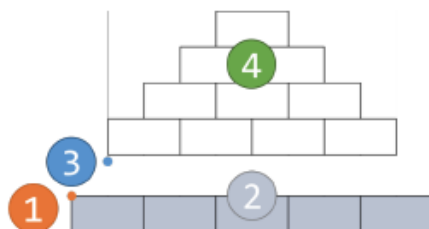
Als je een piramide van $n = 5$ lagen van onderaf begint te tekenen, dan heb je na het tekenen van de onderste laag stenen daarna nog 4 lagen te gaan. Die 4 lagen vormen zelf ook een piramide! Als we na het tekenen van één laag de functie `tekenPiramide` voor $n = 4$ uitvoeren, krijgen we uiteindelijk een piramide van 5 lagen. In stappen (zie figuur 1.51):

- Verplaats (`translate`) de tekenpositie naar het begin van de eerste laag (1)
- Teken de eerste laag stenen van in totaal $n = 5$ (2)
- Verplaats de tekenpositie naar het beginpunt van de bovenliggende laag (3)
- Geef de opdracht om vanaf dat punt een nieuwe piramide te tekenen voor $n-1 = 4$ (4)
- Herhaal dit, zolang de piramide nog niet klaar is, dus zolang $n > 0$.

In figuur 1.50 zie je deze stappen vertaald naar programmeercode. Dit is de code van *voorbeeld 9*. In deze paragraaf gaan we recursie inzetten voor het oplossen van problemen.

```
function draw() {  
  translate(0, height - hoogte);  
  tekenPiramide(aantalLagen);  
}  
  
function tekenPiramide(n) {  
  if (n > 0) {  
    for (var nr=0; nr<n; nr++) {  
      rect(nr * breedte, 0, breedte, hoogte);  
    }  
    translate(breedte / 2, -hoogte);  
    n--;  
    tekenPiramide(n);  
  }  
}
```

FIGUUR 1.50



FIGUUR 1.51

★ Opdracht 36 piramide

In opdracht 35 heb je een piramide geprogrammeerd. In *voorbeeld 9* wordt dezelfde piramide geprogrammeerd, maar nu met recursie.

186. Open *H1O36.js* in jouw editor. Bekijk het resultaat in de browser.

187. Zorg dat een piramide met 10 lagen wordt getekend.

Hoewel de functie `tekenPiramide` zichzelf aanroept, gaat het programma niet oneindig lang door. Dit komt omdat de programmeur een voorwaarde heeft ingebouwd.

188. Wanneer stopt de functie met de uitvoer? Ofwel: wat is de stopconditie?

Als we het aantal lagen groter maken, neemt de grootte van het canvas ook toe. Het canvas is op dit moment $10 \times 90 = 900$ pixels breed en 450 pixels hoog. We willen bereiken dat het canvas altijd een grootte van 900×450 pixels heeft en dat het programma zelf uitrekt hoe breed en hoog de stenen dan kunnen worden.

189. Pas de code aan, zodat bovenstaand doel wordt bereikt.

190. Maak een piramide met 100 lagen.



FIGUUR 1.52

★ Opdracht 37 Droste-effect

In figuur 1.53 zie je een wereldberoemd cacaoblik van de Nederlandse firma Droste. Op het cacaoblik staat een serveerster die een dienblad vasthoudt met daarop een cacaoblik met daarop een serveerster die een dienblad vasthoudt met een cacaoblik met daarop... etc. Deze eindeloze herhaling wordt het Droste-effect genoemd. Het is een voorbeeld van recursie.

191. Open *H1O37.js* in jouw editor. Bekijk het resultaat in de browser. Je ziet een kamer met een deur. Aan de wand hangt een groot zwart schilderij.

We willen dat op het schilderij het beeld van de kamer wordt herhaald. Dit beeld wordt gemaakt met de functie `tekenKamer`. Als we hier recursie willen toepassen, moeten we zorgen dat de functie `tekenKamer` zichzelf aanroept.

192. Voeg aan het eind van de functie de volgende regel toe:
`tekenKamer(0.5);`

Als het goed is zie je nu het Droste-effect.

193. Gaat dit programma oneindig door? Of is er een stopconditie? Zo ja, onder welke voorwaarde stopt dit programma?

We kunnen nu onderdelen toevoegen aan de kamer en kleuren aanpassen, zoals het voorbeeld in figuur 1.54.

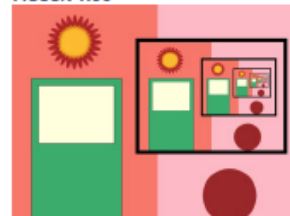
194. Voeg een grote bal toe onder het schilderij. Bekijk het resultaat.

195. Pas de kleuren van de kamer aan, zodat ze passen bij jouw smaak.

196. Voeg minimaal één ander object toe aan de ruimte. In figuur 1.54 hebben wij gekozen voor de bloem die we eerder dit hoofdstuk hebt getekend, maar je kunt ook kiezen voor iets geheel nieuws.



FIGUUR 1.53



FIGUUR 1.54

VB hoofdstuk I: raak of mis?

Opdracht 23 `keysDown`: raak het andere blokje

We hebben al kennis gemaakt met een standaard P5-functie die reageert als er wordt geklikt (bij `mouseIsPressed`). Er zijn ook functies die reageren op het toetsenbord. In deze opgave kijken we naar `keyIsDown`. De functie `mouseIsPressed` heeft geen parameter, want er is maar één muis), maar aan bij `keyIsDown` moet je een argument meegeven. In *H1O23.js* hebben we al een beginnetje gemaakt.

129. Open *H1O23.js* in jouw *editor*. Bekijk het resultaat in de *browser*. Druk op de pijltjestoetsen. Welke richtingen werken?

130. Het blokje blijft in het canvas, door regel 23:

```
y = constrain(y, 0, height - 100);
```

Verklaar het gebruik van `height - 100` in deze functie.

131. Pas de code aan, zodat het blokje ook naar links en rechts kan bewegen (met de bijbehorende pijltjestoetsen).

HINT: Gebruik **LEFT** en **RIGHT**.

132. Het blokje kan nu wel links en rechts het canvas verlaten. Blokkeer dit met de functie `constrain`.

Het was je vast al opgevallen dat het rechterblokje fel groen (*chartreuse*) kleurt, als het vierkantje zich op een bepaalde hoogte bevindt zoals in figuur 1.32. Om precies te zijn is gezorgd dat het blokje kleurt als de hoogtes van beide blokjes elkaar (deels) overlappen. Dit is bereikt met `if (y >= 75 && y <= 225)`.

133. Leg uit waarom hier voor de waarden `75` en `225` is gekozen.

134. Waarom staat er in deze coderegel `&&` en niet `||`? Voorspel wat er gebeurt als we dit aanpassen. Controleer jouw voorspelling. Vergeet niet om daarna de `&&` weer terug te zetten.

135. Breid de code uit, zodat het rechterblokje groen kleurt wanneer het geraakt wordt door het vierkant.



FIGUUR 1.32

2.6 zelf objecten maken

In de vorige paragrafen heb je kennis gemaakt met de bijzondere objecten afbeelding en lijst. Ze zijn bijzonder, omdat hun werking afwijkt van andere objecten, maar belangrijk voor nu is dat we hebben gezien dat ze eigenschappen hebben en dat je ze kunt vragen om een handeling uit te voeren (figuur 2.29).

AFBEELDINGEN

```
// attributen: eigenschappen
spriteJos.width;
spriteJos.height;

// methodes: handelingen
spriteJos.hide();
```

LIJSTEN (ARRAYS)

```
// attributen: eigenschappen
klas.length;
klas[2];

// methodes: handelingen
klas.sort();
klas.push("Trent");
```

OBJECT kever

```
// attributen: eigenschappen
kever.x;
kever.y;

// methodes: handelingen
kever.beweeg();
```

FIGUUR 2.29

Het maken van programma's met objecten wordt object-georiënteerd programmeren genoemd. Het is een bepaalde tactiek van programmeren, of programmeer-paradigma, waarbij je eigenschappen en handelingen koppelt onder één noemer: het object. We leggen het uit met een voorbeeld:

In *voorbeeld 10* (zie ook § 2.2) laten we een kever over het canvas bewegen. Er is een variabele `kever` voor de bijbehorende sprite en er zijn variabelen `kever.x` en `kever.y` om zijn positie mee te bepalen. Daarnaast is er code om de kever te laten bewegen.

Eigenlijk is dit onhandig, want dit zijn allemaal losse gegevens, terwijl ze eigenlijk bij elkaar horen. Net zoals bij de afbeeldingen en lijsten, willen we de kever vragen om informatie over zichzelf te geven of om iets voor ons te doen, zoals in het rechterblok van figuur 2.29. We willen van de kever een object maken.

In *voorbeeld 15* is de kever van *voorbeeld 10* nogmaals te zien, maar nu object-georiënteerd. De declaratie van ons object `kever` zie je in figuur 2.30.

Een eigenschap van een object heet attribuut. In dit voorbeeld zijn er drie attributen: `x`, `y` en `sprite`. De eerste twee hebben we meteen een waarde gegeven, maar `sprite` blijft nog even leeg (omdat we daar straks in de `preload` een afbeelding inladen). Als je een attribuut nog niet meteen een waarde meegeeft, kun je dat aangeven met `null`. Let goed op het gebruik van `{ }`, `:` en `,` bij het maken van een object. Dat komt heel precies!

Een handeling van een object heet methode. Als we de kever vragen om te bewegen via de methode `beweeg()`, dan worden drie regels uitgevoerd. Deze regels gebruiken de attributen `x`, `y` en `sprite`, maar wel steeds voorafgegaan door `this`.

Het gebruik van `this` is even wennen. Het is een verwijzing naar de eigenaar van een attribuut of methode; in ons geval `kever`. Het benadrukt dat het gaat om eigenschappen en handelingen van *dit* object. Dat gebruiken van `this` doe je alleen bij het maken van een object. In het hoofdprogramma gebruik je de naam die je aan het object hebt gegeven. Hier een voorbeeld voor het gebruik van een attribuut:

```
text("De kever bevindt zich op x-positie" + kever.x,0,0);
```

en een voorbeeld voor het gebruik van een methode:

```
kever.beweeg();
```

```
var kever = {
  // attribuut
  x: 100,
  y: 150,
  sprite: null,

  // methode
  beweeg() {
    this.x += random(-5,5);
    this.y += random(-5,5);
    image(this.sprite,this.x,this.y);
  }
};
```

FIGUUR 2.30



Opdracht 17 werken met objecten

75. Open `H2O17.js` in jouw editor. Dit is de code van *voorbeeld 15* met enkele kleine aanpassingen. Bekijk het resultaat in de browser.
76. De tekst onderaan verandert op dit moment niet mee als de positie van de kever verandert. Pas de code aan, zodat de actuele waarden van `x` en `y` worden getoond.
77. Voeg een attribuut naam toe aan het object `kever` en geef als waarde een zelfbedachte naam mee. LET OP: omdat een naam een tekst is, moet deze tussen aanhalingstekens.
78. Pas de getoonde tekst aan zodat in plaats van "Het object kever" de door jou gekozen naam verschijnt. Gebruik de code `kever.naam` voor het tonen van de naam.

Opdracht 18 Jos als object

79. Open `H2O18.js` in jouw editor. Bekijk het resultaat in de browser. Je ziet de getekende versie van Jos uit hoofdstuk 1.

In hoofdstuk 1 werd Jos getekend met een zelfgemaakte functie. De coderegels staan nu als methode teken binnen het object `jos`.

80. Hoeveel attributen heeft het object `jos`?
81. Wat is op dit moment de waarde van het attribuut `jos.x`?
82. De methode teken heeft nu als parameter `muissetiex`. De bedoeling is dat Jos ook echt gaat meebewegen met de muis. Pas de regel `jos.teken(500)` aan, zodat Jos reageert op de x-positie van de muis.

We willen dat Jos groter wordt als hij naar rechts beweegt en kleiner als hij naar links beweegt.

83. Voeg de volgende regel toe aan teken in regel 9: `this.schaal = this.x / (0.25*width);`
84. Zorg dat in de tekst bovenaan behalve de x-positie ook de schaal van Jos wordt getoond.
85. Pas de code aan, zodat Jos ook reageert op de y-positie van de muis. Doe de volgende stappen:
 - Zorg dat de methode teken een extra parameter `muissetiex` krijgt.
 - Gebruik de functie `constrain` om te zorgen dat y (van Jos) tussen de 100 en 150 blijft.
 - Zorg dat bij het aanroepen van de methode teken de y-positie van de muis als parameter wordt meegegeven.

Opdracht 19 overloper IV: het raster als object

In deze opgave gaan we het paradigma van object-georiënteerd programmeren toepassen op het spel *overloper* dat als rode draad door dit hoofdstuk loopt. We beginnen met het raster.

86. Open `H2O19.js` in jouw editor en bestudeer de code. Dit is qua werking het eindresultaat van *overloper III*.
87. Maak een object raster met de attributen `aantalRijen`, `aantalKolommen` en `celGrootte` en geef ze achtereenvolgens de waarden `6`, `6` en `null` mee.
88. Voeg de methode `berekenCelGrootte()` toe aan het object raster volgens het voorbeeld in figuur 2.32.
89. Om de `celGrootte` te berekenen, moet deze methode nog wel worden aangeroepen. Vraag binnen de `setup` het object raster om de methode `berekenCelGrootte()` uit te voeren.
90. De oude variabele `celGrootte` willen we niet meer gebruiken. Pas alle coderegels die `celGrootte` gebruiken aan, zodat ze het attribuut `raster.celGrootte` gebruiken.
91. Voeg de methode `teken()` toe aan het object raster, zodat de regel `raster.teken()` zorgt voor het tekenen van het raster. Gebruik de coderegels uit de oude functie `tekenRaster()` als basis.
92. Welke oude coderegels kun je nu allemaal verwijderen?
93. Voeg de regel `raster.teken()` toe aan de `draw`. Wordt het raster nog steeds getekend?

```
berekenCelGrootte() {
  this.celGrootte =
    width/this.aantalKolommen;
}
```

FIGUUR 2.32



obfuscator: XIV Galgje IV

beginnen we nog een aantal verbeteringen toe aan het spel *Galgje*.

- Een timer toont het aantal resterende beurten tenzij je het woord geraden hebt.
 - Als het spel afgelopen is, kun je door op de spatiebalk te drukken een nieuw spel beginnen. Gedurende het spel mag het drukken op de spatiebalk geen effect hebben.
 - Het spel bevat een array met woorden waaruit willekeurig een opgave wordt gekozen.
27. Bekijk *OBFI4*. Probeer het spel uit en stel vast dat het aan bovenstaande eisen voldoet.
 28. Open *H3O5.js* in jouw editor en bestudeer de code. Breid de code uit, zodat aan de eerste eis (resterende beurten) wordt voldaan.
 29. De invoer van de speler wordt in eerste instantie verwerkt in *keyTyped*. Breid deze uit met een voorwaarde, zodanig dat *setup* (voor een nieuw spel) opnieuw wordt uitgevoerd op het moment dat op de spatiebalk wordt gedrukt. Natuurlijk mag dat alleen aan het eind van en niet tijdens een spell!
 30. Pas de constructor van *Galgje* aan zodat het te raden woord willekeurig wordt gekozen uit een array met woorden.

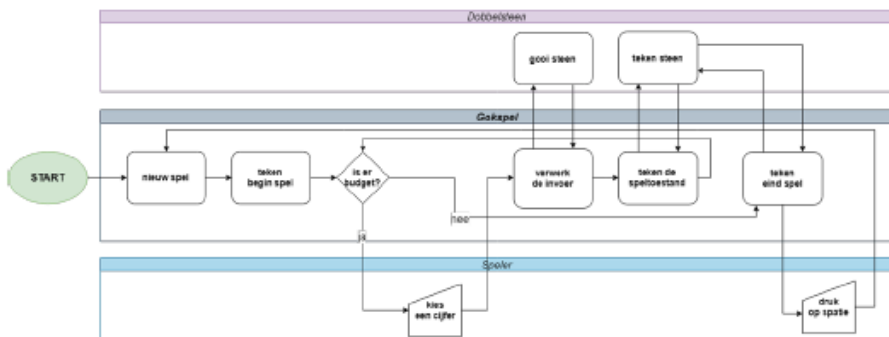
Opdracht 6 Gokspel

In het vorige hoofdstuk hebben we een klasse *Dobbelsteen* gemaakt die we gaan inzetten voor met slechts een paar eenvoudige spelregels:

- De speler begint met een budget van 5.
- De speler raadt hoeveel ogen er zal worden gegooid. Vervolgens gooit het spel de dobbelsteen.
- Als de speler het goed geraden heeft, krijgt hij er 5 punten bij. Is het fout, dan gaat er 1 punt af.
- Je bent af als er geen budget meer over is. In dat geval start je met de spatiebalk een nieuw spel.

31. Open *H3O6.js* in jouw editor en speel een aantal keren *Gokspel*.
32. Welke aspecten van dit spel zouden het aantrekkelijk kunnen maken om te spelen?
33. Reageerde je anders op het spel op het moment dat je een worp goed had voorspeld? Waarom?
34. Welk element van het spel zorgt ervoor dat een speler door blijft spelen?

Een programmeur heeft vooraf onderstaande flowchart bedacht met daarin drie (klassen van) objecten:



FIGUUR 3.9

35. Welke overeenkomsten zie je tussen deze flowchart en die van het spel *Galgje* in figuur 3.2?
36. Open *H3O6.js* in jouw editor. Op welke punten wijkt de code af van de flowchart in figuur 3.9?
37. Een nieuw spel beginnen mag alleen als het vorige spel is afgelopen. Met welke code wordt voorkomen dat een speler middenin een spel een nieuw spel kan beginnen met de spatiebalk?
38. Regel 101 luidt `this.speler.budget -= this.straftout`; Onder welke voorwaarde (-n) wordt deze coderegel uitgevoerd? Beschrijf de betekenis van deze regel in spreektaal.

Opdracht 7 Codekraker I

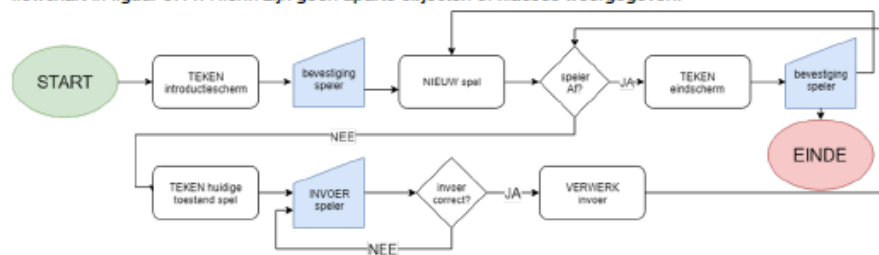
In figuur 3.10 zie je een screenshot uit het spel *Codekraker*.



FIGUUR 3.10

39. Open *H3O7.js* in de browser (het is niet nodig om de code in jouw editor te openen) en probeer het spel te spelen. Probeer de precieze spelregels te ontdekken door het spel te spelen.
 40. Beschrijf de spelregels stapsgewijs. De bedoeling is dat je het zo opschrijft, dat een goede programmeur op basis van jouw omschrijving het spel zelf kan programmeren.
- Zie de vorige opdracht voor een voorbeeld van spelregels.

Het spel *Codekraker* heeft qua structuur grote overeenkomsten met *Gokspel*: Er is een startscherm dat door een actie van de speler verdwijnt. Daarna moet je iets raden. Elke keer dat je iets raadt, controleert het spel of je het goed hebt gedaan. Als je af bent, verschijnt er een eindscherm en kun je ervoor kiezen om nogmaals te spelen. De twee spellen zijn te beschrijven met de algemene flowchart in figuur 3.11. Hierin zijn geen aparte objecten of klassen weergegeven.



FIGUUR 3.11

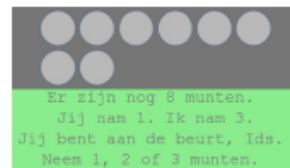
41. Noem nog twee bestaande spellen die te programmeren zijn volgens het schema van figuur 3.11.

Qua spelervaring lijkt *Codekraker* meer op *Galgje* dan op *Gokspel*, omdat er niet alleen een factor *geluk* is maar je ook na kunt (moet!) denken over een slimme strategie.

42. Vind je dat de mogelijkheid van inzet van een strategie een spel aantrekkelijker maakt? Leg uit.
43. Welke strategieën kun je bij het spel *Codekraker* inzetten?
44. Welke strategieën kun je bij het spel *Galgje* inzetten?
45. Beschrijf de factor *geluk* bij *Codekraker* en bij *Galgje*.

Opdracht 8 Nim

46. Open *H3O8.js* in de browser en speel Nim (figuur 3.12). De spelregels worden uitgelegd op het openingsscherm.



FIGUUR 3.12

Deze versie van Nim volgt de flowchart van figuur 3.11.

47. Open *H3O8.js* in jouw editor en zoek de methode die het proces *invoer correct?* uit de flowchart afhandelt. Hoe heet de bijbehorende methode? Wat wordt er allemaal gecontroleerd om vast te stellen dat de invoer correct is?
48. Heeft de programmeur de munten hier als object geprogrammeerd? Hoe zie je dat?

Als je het spel een paar keer hebt gespeeld, kom je erachter dat je altijd verliest van de computer. Dat komt omdat er een optimale strategie voor het spel is: als je de truc kent, wint de tweede speler altijd.

49. Bij welk proces in de flowchart kiest de computer hoeveel munten hij zelf pakt?
50. Zoek de bijbehorende methode in de Javascript-code. Wat is de optimale strategie bij dit spel?

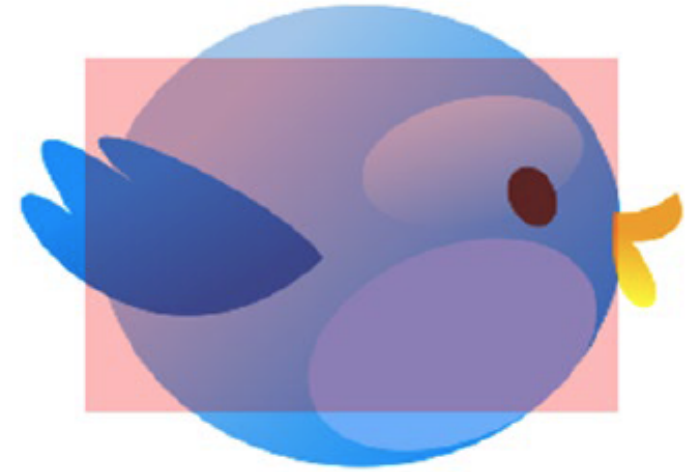
VB hoofdstuk 3: events

Opdracht 18 Bluebird I

Je hebt vast weleens *Flappy Bird* gespeeld. In deze opgave maken we de variant *Bluebird*.

106. Open *H3O18.js* in de browser en jouw *editor* en speel *Bluebird*.

Dit spel maakt gebruik van afbeeldingen. Dat maakt het lastiger om vast te stellen of de vogel een obstakel raakt dan met een cirkel of rechthoek. De programmeur heeft dit opgelost door een rechthoekig gebied te maken (figuur 3.23) waarmee wordt bepaald of het raak is. Vanaf de rand van de afbeelding wordt een marge aangehouden, zodat je alleen af bent als je een obstakel met het gekleurde deel raakt



FIGUUR 3.23

- 107. Bestudeer de methode `raakt(vogel)` van de klasse `Obstacle`. Klopt de reeks met voorwaarden?
- 108. Zorg dat het spel de marge nog steeds gebruikt, maar het rode vlak niet meer laat zien.
- 109. Pas het spel aan zodat het met het toetsenbord kan worden gespeeld. Gebruik de spatiebalk om te vliegen en Enter om het spel te beginnen.

Hoe gebruik je het lesmateriaal?

Cloud9 IDE (*integrated development environment*)

Opening workspaces (including access to databases and app frameworks) will be disabled on June 30, 2019. Cloud9 will stop working on December 31, 2019. [Learn more](#)

Cloud9 File Edit Find View Goto Run Tools Window Support Preview Stop

Workspace

- slo-games
 - SLOsite
 - CSS
 - DOCENT
 - images
 - JS
 - H1
 - H2
 - H3
 - HGA
 - HOBf
 - HVB
 - P5
 - prism.js
 - SLO.js
 - SLOBF.js
 - test.js
 - bekijkJS.html
 - favicon.ico
 - index.html

Commands

Navigate

Commands

C1.js

```
1 function setup() {  
2   var myCanvas = createCanvas(450,450);  
3   background('silver');  
4   myCanvas.parent('processing');  
5   //noLoop();  
6 }  
7  
8 function draw() {  
9   noStroke();  
10  fill('steelblue');  
11  ellipse(0,0,800);  
12  fill('deepskyblue');  
13  ellipse(450,450,400);  
14 }
```

[B] /SLOsite/bekij x

/SLOsite/bekijkJS.html

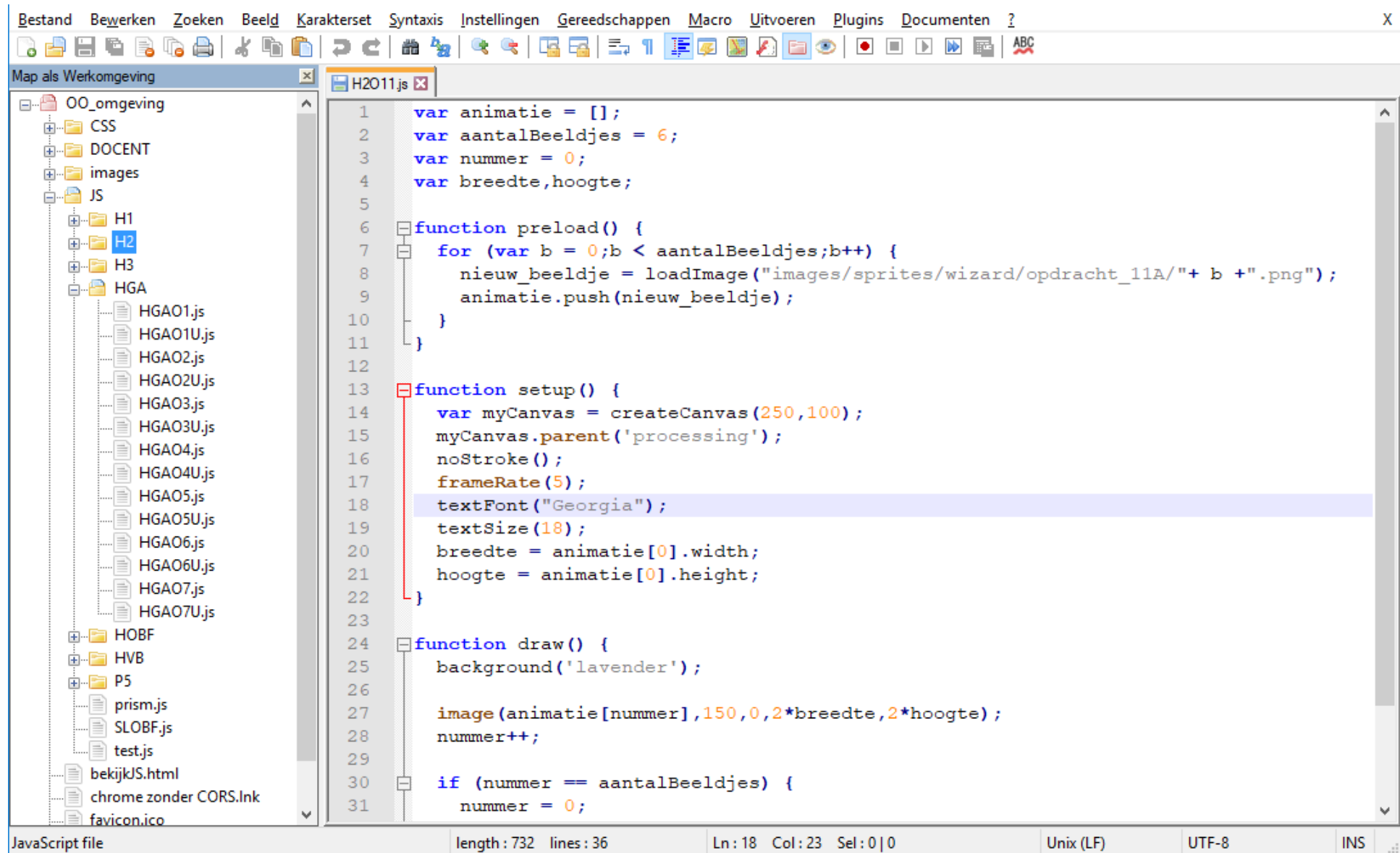
Browser

Pas regel 12 van **bekijkJS.html** om van opdracht te wisselen.



Collaborate Outline Debugger

Notepad++ (lokaal / offline werken)



CORS: lokale beperking bij afbeeldingen

Wat is CORS?

CORS (Cross-Origin Resource Sharing) is een HTTP-functie waarmee een webtoepassing die wordt uitgevoerd in een bepaald domein te krijgen tot bronnen in een ander domein. Als u wilt verminderen de kans op aanvallen via cross-site scripting, alle moderne webbrowsers implementeren van een beveiligingsbeperking bekend als [beleid voor zelfde oorsprong](#). Dit voorkomt dat een webpagina van aanroepen van API's in een ander domein. CORS biedt een veilige manier om toe te staan een bron (het domein van oorsprong) API's aanroepen in een andere oorsprong.



Opgave 1 (H2) | Kies

VB

H1

H2

H3

OBF

GA

01

02

03

04

05

06

07

08

09

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

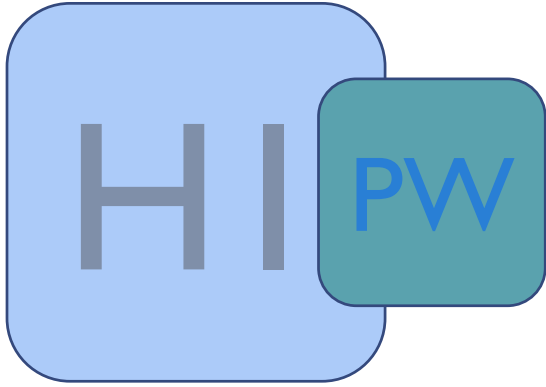
41

Toon Javascript-Code

Toon Uitwerkingen

Loading...

module games maken en ervaren



- kennismaken met JS en P5
- basisprincipes programmeren
- loop-functie
- (eerste) interactie
- (verdieping) recursie

- **niet** while
- **niet** case

Opfrissen & P5

- Hoe heb je het programmeerdeel uit het kernprogramma aangepakt?
- Wat zit er in je methode?

Afhankelijk van deze vragen kan HI snel worden afgesloten. Er zijn proefwerken op zowel havo- als vwo-niveau beschikbaar die formatief of summatief kunnen worden ingezet.

- ☐ Opfrissen van de stof is sowieso noodzakelijk
- ☐ Kennismaken met P5 en de leeromgeving hoeft niet veel tijd te kosten
- ☐ VWO: neem je het deelparadigma recursie mee?

module games maken en ervaren

H2

- standaard objecten (array, afbeelding)
- sprites & spritesheets (verdieping)
- zelf objecten maken
- een object dat antwoordt
- klasse van objecten
- array van objecten (van één klasse)

- **niet** overerving

OO Paradigma

- veel keuze: je hoeft niet alles te doen
- basisopdrachten voor havo, steropdrachten voor de betere leerling
- ofbuscator-opdrachten als extra uitdaging
- Pak je alleen de rode draad *Overloper*?

Er is ruimte om als docent of leerling een eigen route te kiezen. Kies wel:

- ☐ bestaande objecten
- ☐ zelf objecten maken
- ☐ klassen
- ☐ array van objecten als je H3 wilt doen

gamification is
using game-based mechanics,
aesthetics, and game-thinking to **engage**
people, motivate action, promote
learning, and solve problems.

Gamification 4 | Kies

[VB](#)[H1](#)[H2](#)[H3](#)[OBF](#)[GA](#)[01](#)[02](#)[03](#)[04](#)[05](#)[06](#)[07](#)[08](#)[09](#)[10](#)[11](#)[12](#)[13](#)[14](#)[15](#)

A: $x=412$ $v=2$



B: $x=211$ $v=2$

[Toon Javascript-Code](#)[Toon Uitwerkingen](#)

- per paragraaf een *level*
- bewijzen dat je het kunt door het probleem op te lossen
- eigen tempo
- differentiatie
- competitie
- belonging

WHY DOES GAMIFICATION WORK IN EDUCATION?

Gamification helps people learn by doing, which ultimately improves processes and outcomes (Shute & Ventura, 2013). Gamification provides learners with the ability to learn on their own time and at their own pace. Gamification also allows learners to follow their progress, providing autonomous learning (Klopfer et al., 2009). Participants enjoy the freedom to fail while experimenting in a nonthreatening environment (Cook, 2013; Lazzaro, 2004). Learners can experience emotions such as frustration, wonder, mystery, and amusement, each providing a personal connection to the game or others playing the game (Lazzaro, 2004).

Importance of Gamification in Increasing Learning

Stacey Brull, DNP, RN, NE-BC; and Susan Finlayson, DNP, RN, NE-BC

GAMIFICATION MECHANICS

In order for gamification to work, the literature suggests that specific game mechanics need to be in place. Game mechanics include badges, points, challenges, rewards, leaderboards, and levels (Hamari, Koivisto, & Sarsa, 2014; Hanus & Fox, 2015). Using game mechanics and other types of gaming strategies allows learners to solve problems in an engaging and fun way (Bruder, 2015). Using game mechanics can increase the average retention rate of information up to 10 times higher than that resulting from lecture (Cook, 2013). Experts in gamification caution educators to not merely add a gaming mechanic to a course and expect positive outcomes (Farber, 2015). Game mechanics need to be combined with achievable goals, rules, voluntary participation, and feedback to work (McGonigal, 2011). Below are brief descriptions of four of the more popular mechanics seen in gamification.

Importance of Gamification in Increasing Learning

Stacey Brull, DNP, RN, NE-BC; and Susan Finlayson, DNP, RN, NE-BC

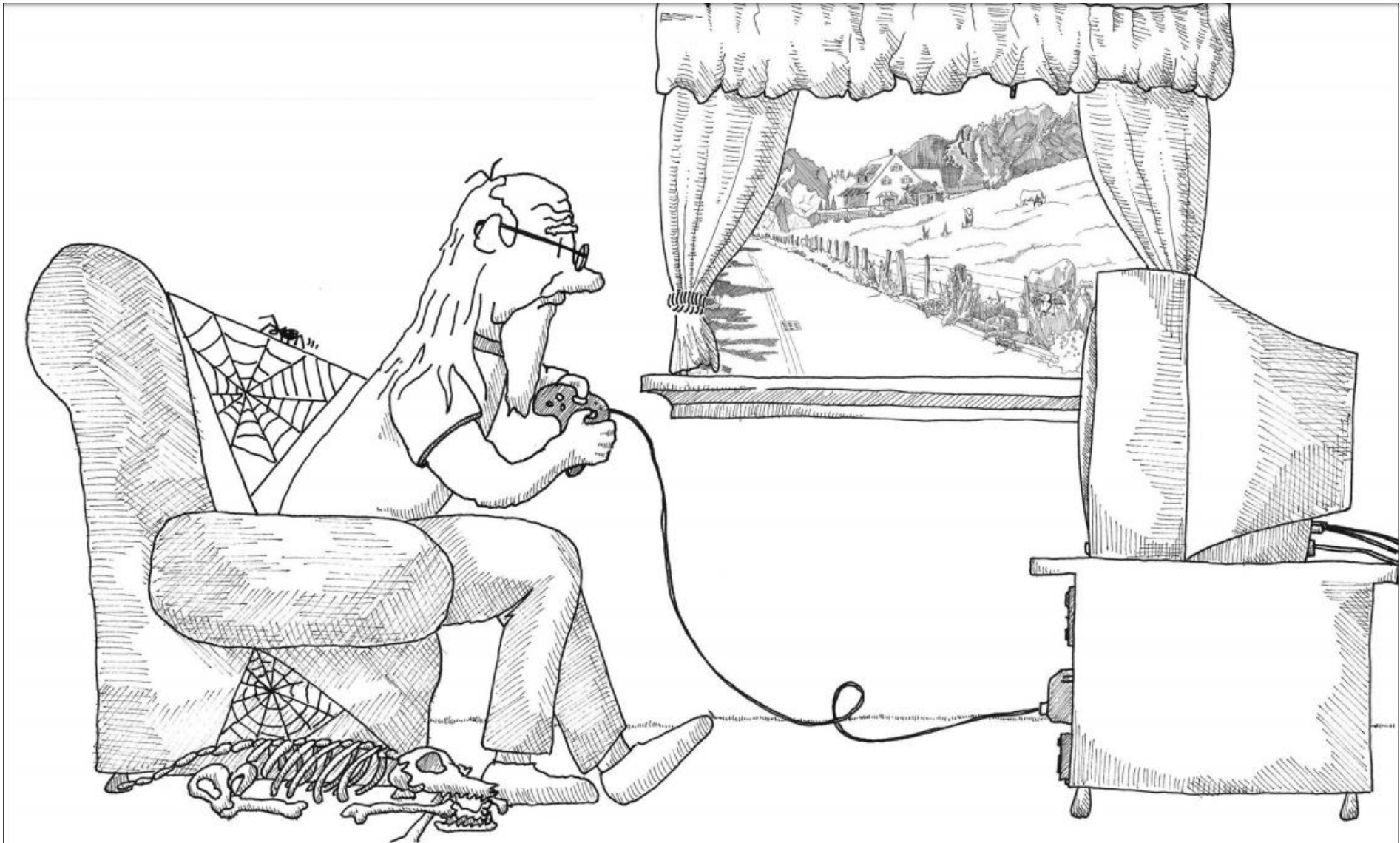
gamification leaderboard & stickers

Naam	Score	Aantal	X	A	B	C	D	E	F	G	H	I	J	K	L	M	N
Aniek	10	5															
Anne-Floor	14	7															
Cécile																	
Daan	9	3															
Daniëlle	10	5															
David	9	4															
Eline	12	6															
Emma	9	4															
Esther	7	4															
Evert Jan																	
Folkert	18	8															
Hannah	8	4															
Jasper	15	6															
Laura	11	5															
Luca	7	4															
Mart	7	4															
Matilda																	
Merlijn																	
Mia	8	4															
Nikos	8	5															
Noah	12	6															
Ole	11	5															
Parisa	10	5															
Thomas	5	4															
Tosca																	
William	5	3															
Yaël	12	5															

goud

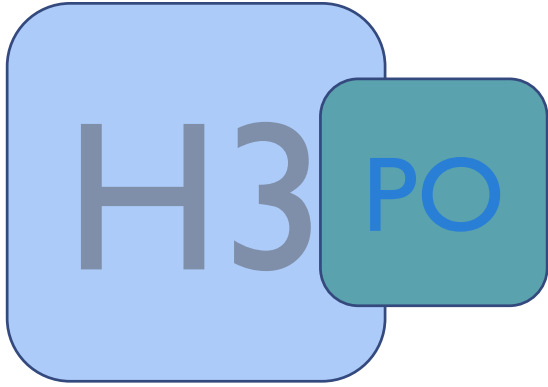
zilver

brons



*And fun is a neurochemical reward to encourage us
to keep trying.*

module games maken en ervaren



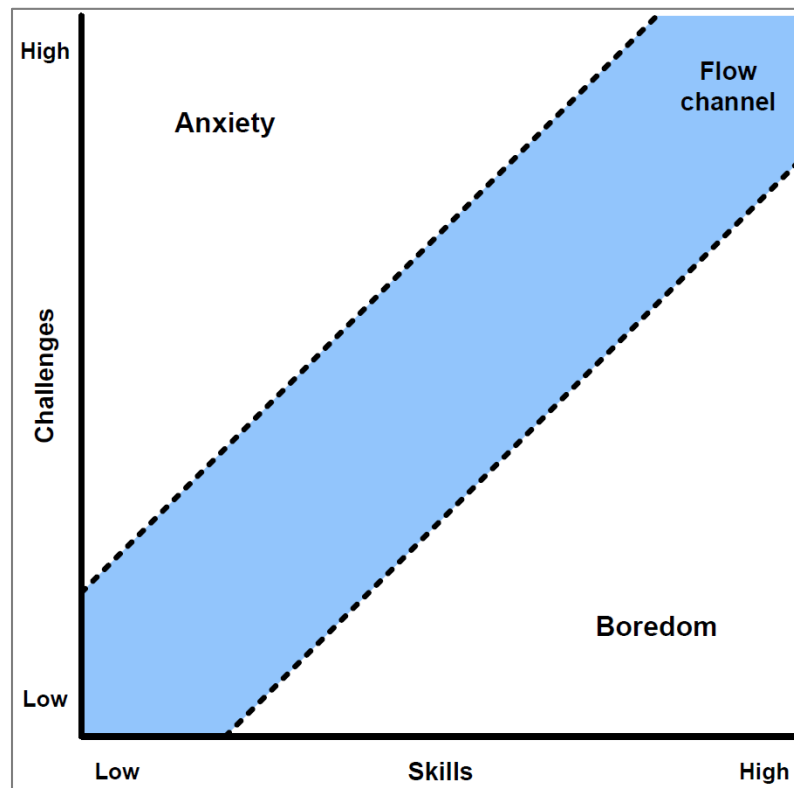
- structuur van een spel: flowcharts
- object spel
- events, acties, gebeurtenissen
- design patterns
- spelregels
- levels
- tijd
- geluid

Structuur & Ervaring

- veel keuze: je hoeft niet alles te doen
- filter op UX-opdrachten of OO-opdrachten
- hoeveel nadruk wil je leggen op flowcharts?
- laat leerlingen *shoppen* op basis van hun PO-plan
- programmeer-deel lastig voor havo

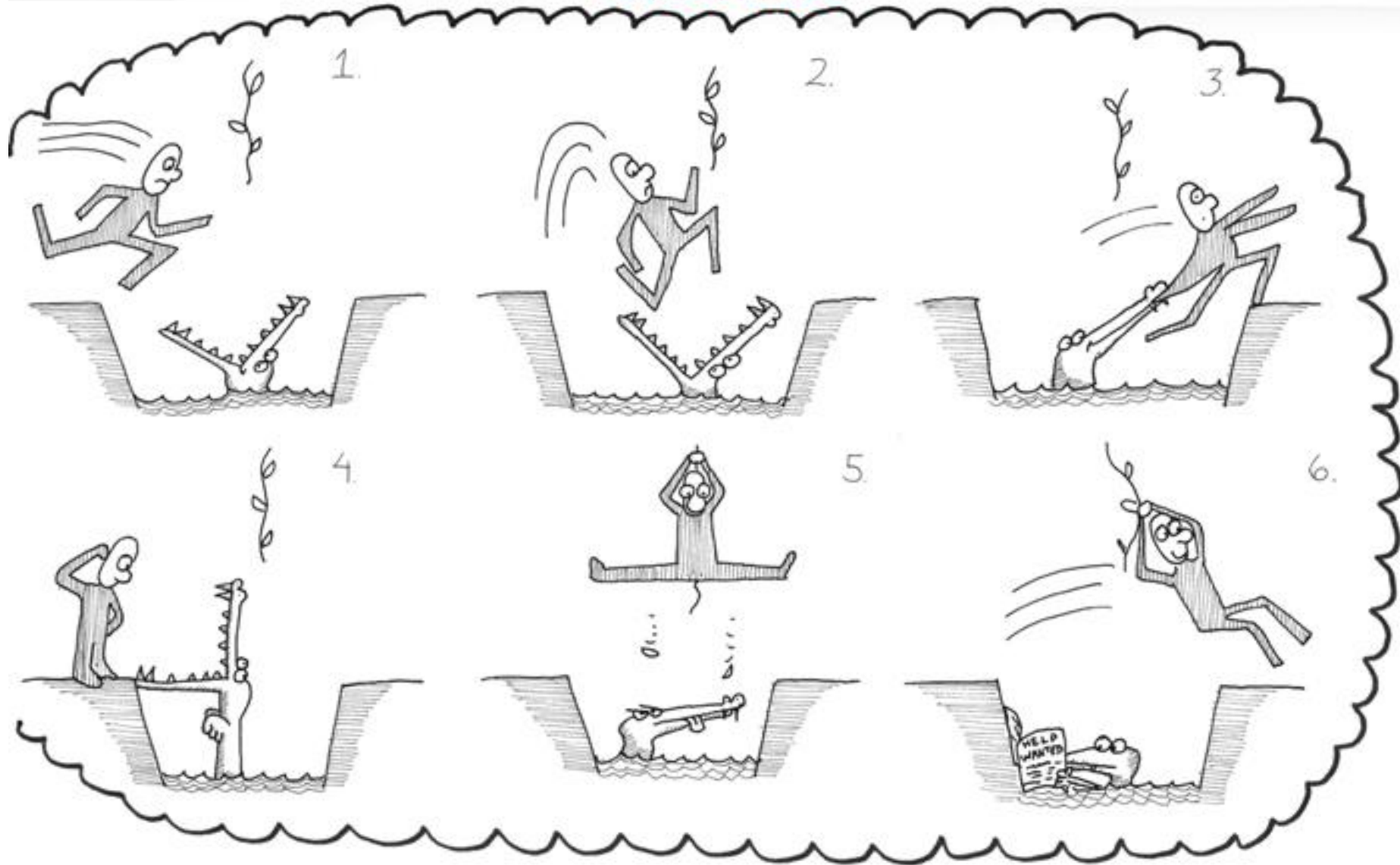
Er is ruimte om als docent of leerling een eigen route te kiezen. Kies wel:

- ☐ het object *spel*
- ☐ design pattern: houvast voor het PO
- ☐ *game*-elementen: doel, beloning, levels, interactie



A game can facilitate the flow experience if the challenges that the game offers are **up to par** with the skills of the player. If a game does not provide enough challenge, the player eventually gets bored. If a game provides too much challenge, the player might experience anxiety or quit after endless defeats. **If the challenges of a game are equal to the player's skills, the player enters the flow channel** (Csikszentmihalyi, 1990)

Playing games exercises your brain.



oplopende moeilijkheidsgraad

- Hoe neemt de moeilijkheidsgraad hier toe?
- Welk gevolg heeft dat voor de score?
Moet dat anders?
- Welke rol speelt geluk (*random*) hier?
- Hoe verandert het gedrag / de emotie van de speler als de zoektijd wordt beperkt?
- Hoe kun je de zoektijd gebruiken om een oplopende moeilijkheidsgraad te bereiken?
- Zie je andere mogelijkheden om verandering aan te brengen bij een volgend level?
- Is het mogelijk om vals te spelen?
- etc.



keuzes en dilemma's
ambitie versus realiteit

beperkt aantal JS- & P5-functies

soms zou je het zelf toch anders doen

het is geen uitgebreide JS-cursus

Leesbaarheid is belangrijk
ontwikkeling in abstractie idem

soms zou je het zelf toch anders doen

slimmer en korter gaat vaak niet samen met duidelijk
bewust niet meteen *de beste code*

C2

C3

Wanneer is het ○○ (genoeg)?

spanningsveld tussen
correctheid en volledigheid &
begrijpelijkheid en haalbaarheid

~~hoe strenger hoe beter~~

Wanneer is het ○○ (genoeg)?

geen getters en setters

- wel **waarde = 0** en **waarde == 0**,
maar niet **waarde === 0**
- geen nadruk op lokale en globale variabelen
- wel parameters / argumenten / attributen
- wel *derived properties* als dat gemak geeft:
this.straal = this.diameter / 2
- wel expliciet **this.leeg = null**; (i.p.v. **this.leeg**;))
- wel event-handlers

standaard gebruik van een **klasse**

ook als er maar één instantie wordt gemaakt
(maar wel aangeleerd in stapjes)

```

var kever;
var keverX;
var keverY;

function preload() {
  kever = loadImage("kever.png");
}

function setup() {
  keverX = 150;
  keverY = 100;
}

function draw() {
  keverX += random(-5,5);
  keverY += random(-5,5);
  image(kever,keverX,keverY);
}

```

VB10

```

var kever = {
  x: 100,
  y: 150,
  sprite: null,

  beweeg() {
    this.x += random(-5,5);
    this.y += random(-5,5);
    image(this.sprite,kever.x,kever.y);
  }
};

function preload() {
  kever.sprite =
    loadImage("kever.png");
}

function draw() {
  kever.beweeg();
}

```

VB15

```

class Kever {
  constructor(x,y,s) {
    this.x = x;
    this.y = y;
    this.sprite = s;
  }

  beweeg() {
    this.x += random(-5,5);
    this.y += random(-5,5);
    image(this.sprite,kever.x,kever.y);
  }
};

function preload() {
  sprite = loadImage("kever.png");
}

function setup() {
  kevin = new Kever(100,150,sprite);
}

function draw() {
  kevin.beweeg();
}

```

Wanneer is het ○○ (genoeg)?

abstractie, objecten, klassen,
spel.speler.doeZet(),
this.wordtGeraakt(vijand)

Wanneer is het ○○ (genoeg)?

geen overerving!

Inheritance

gives you one way to reuse code,
but often it is in the expense of a
maintenance nightmare.

- ANNI RAUTAKOPRA

Discutabel?

Overerving werd te complex

Hij **doet** het toch?

leerlingen zijn resultaatgericht

tijd om zelf te proberen

het lesmateriaal wordt
beschikbaar gemaakt via het SLO

vragen?