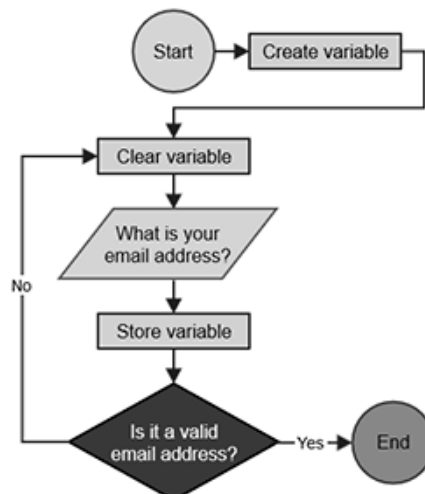


Software met boter, kaas en eieren

Workshop algoritme ontwerpen

Wie verslaat zijn tegenstanders het meest effectief met zijn algoritme?



Software ontwikkelen

Bij het ontwikkelen van software of dat nu voor een game, een boekhoudprogramma, een app voor de telefoon of een website is, onderscheiden we drie verschillende stappen:

1. Maak een algoritme voor het programma
2. Programmeer het algoritme in een programmeertaal
3. Test het programma en los eventuele fouten op

In deze workshop gaan we aan de slag met stap 1: we ontwerpen een algoritme voor een computerprogramma. Het is een belangrijke stap want zonder een goed vooraf ontworpen algoritme kan een computerprogramma of mobiele app nooit goed werken.

Wat is een algoritme?

Een algoritme is een **stappenplan** dat door een programmeur snel en eenduidig omgezet kan worden in code (programmeertaal).



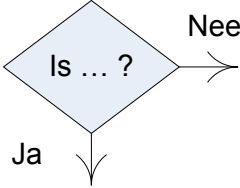
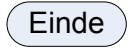

Belangrijke eigenschappen van een algoritme:

- Het is eindig. Het stopt na een bepaald aantal stappen.
- De beginsituatie is duidelijk vastgelegd. Bijvoorbeeld de spelsituatie op het moment dat een speler aan de beurt is.
- Mogelijke eindsituaties zijn duidelijk. Het is duidelijk wat de uitkomst van het algoritme is.

Een goed algoritme levert met zo weinig mogelijk werk een zo goed mogelijk resultaat op.

Hoe maak je een algoritme?

Eén mogelijke weergave van een algoritme is een zogenaamde flowchart. Zo'n flowchart of stroomdiagram bestaat uit de volgende onderdelen:

Element	Betekenis	Toelichting
	Begin	Beginpunt van het algoritme, hier begin je met het doorlopen van het algoritme. Hiervan heb je er precies één per algoritme, en deze staat helemaal bovenaan in je flowchart.
	Activiteit	Een actie die moet worden uitgevoerd: iets wat degene die het algoritme doorloopt moet doen. Doorgaans staan er meerdere van deze blokken in een algoritme.
	Keuze-moment	Afhankelijk van het antwoord op de vraag in dit element (een ja/nee vraag) gaat degene die het algoritme doorloopt door in de 'ja'-richting ofwel de 'nee'-richting. Ook dit element komt doorgaans meer dan eens voor per algoritme.
	Einde	Eindpunt van het algoritme, als je hier bent is je algoritme afgelopen en is een oplossing voor je probleem gevonden. Deze heb je precies één per algoritme en deze staat onderaan.
	Pijl	Voor het verbinden van elementen, altijd eenrichtingsverkeer.

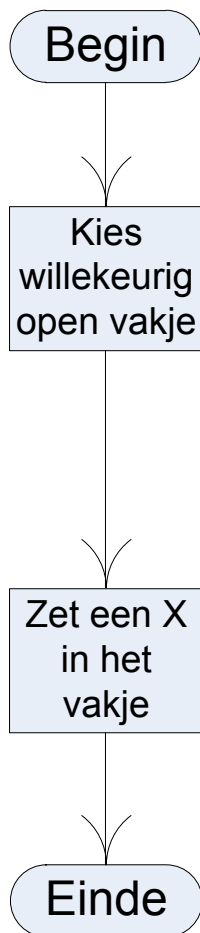
Voorbeeldalgoritme voor boter-kaas-en-eieren

Voor boter-kaas-en-eieren is een algoritme te bedenken voor één speelronde. We gaan er vanuit dat het hier om speler "X" gaat, maar voor speler "O" zou zo'n algoritme er hetzelfde uit zien.

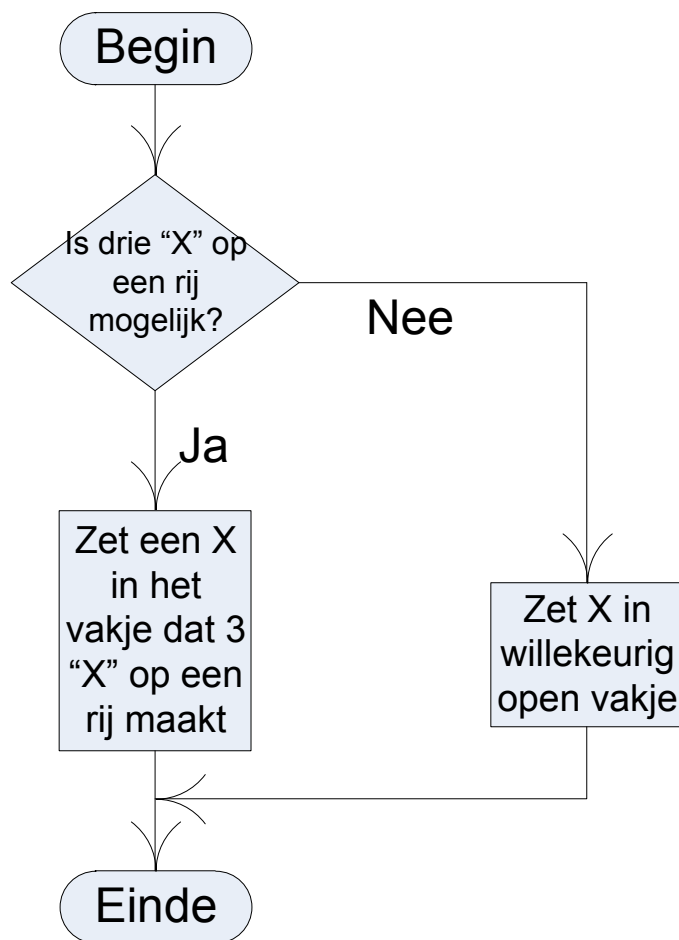
De beginsituatie is: **de speler is aan de beurt** (het speelveld is leeg of al deels gevuld).

De eindsituatie is: **de speler heeft in een leeg vakje zijn teken (kruisje of rondje) geplaatst** OF het spel is verloren/gewonnen/gelijkspel.

Twee verschillende algoritmes zijn:



Algoritme 1



Algoritme 2

Het eerste algoritme zet een kruisje in zomaar een leeg vakje. Erg slim is deze niet, maar de eindsituatie klopt altijd: er wordt in ieder geval iets aangekruist.

Het tweede algoritme kijkt eerst per leeg vakje of de speler "X" kan winnen door dat vakje aan te kruisen, zo niet dan kruist hij een willekeurig vakje aan. Dat algoritme is al wat slimmer.

Opdracht 1

Doel: een goed en winnend algoritme voor boter-kaas-en-eieren ontwerpen. Je werkt eerst samen in een tweetal en later samen in een team van 4 personen.

- 1) **30 minuten:** Maak in tweetallen een algoritme voor boter-kaas-en-eieren dat beter is dan bovenstaande voorbeeldalgoritmes. Controleer of je algoritme aan de volgende eisen voldoet:
 - Het is opgebouwd uit de hierboven beschreven bouwstenen.
 - Het is een **algoritme voor één spelbeurt** (tijdens een spelletje voert dus iedere speler om beurten zijn algoritme uit totdat een speler drie-op-een-rij heeft).
 - Als het algoritme is uitgevoerd **is altijd één leeg vakje gekozen** om een kruisje of rondje (afhankelijk van welke speler je bent) te plaatsen, desnoods kiest je algoritme een willekeurig vakje.

- De betekenis van de tekst in het gemaakte algoritme moet duidelijk en eenduidig zijn (wel duidelijk/eenduidig is bijvoorbeeld “Kan horizontaal, verticaal of diagonaal een rij van drie kruisjes op een rij gemaakt worden?”, niet duidelijk/eenduidig is bijv. “Zet een kruisje op de beste plaats”).
- Het algoritme heeft één duidelijk beginpunt en één duidelijk eindpunt.
- Het is onmogelijk om oneindig in het algoritme te blijven “hangen”.

Opdracht 2 – Het goeroe-algoritme

10 minuten: Speel binnen je groep zo veel mogelijk spelletjes boter-kaas-en-eieren tegen elkaar **volgens je algoritme** (anders is het valsspelen!) waarbij je om-en-om het spelletje begint. Onthoud (even opschrijven) hoeveel spelletjes iedere speler heeft gewonnen. Probeer te achterhalen in welke gevallen je algoritme goed werkt en in welke gevallen je misschien nog niet hebt voorzien. Indien je groep een oneven aantal groepsleden heeft zal je per spelletje moeten rouleren. Optimaliseer je algoritme op basis van je bevindingen tijdens het spelen, en test eventueel je vernieuwde algoritme door tegen een ander groepslid te spelen.

30 minuten: Maak als groep je goeroe-algoritme: het “ultieme” algoritme waarbij je de slimigheden uit de afzonderlijke algoritmes combineert. Maak je goeroe-algoritme op een nieuw vel papier.

Opdracht 3 – Competitie

Iemand tekent een competitie op het bord waarin elk team tweemaal tegen elk ander team speelt (eenmaal als beginner, eenmaal dat de ander begint) en eenmaal tegen zichzelf. Vul het volgende schema klassikaal in (op het whiteboard of digitaal).

LET OP: Beide teams leveren hun algoritme aan, maar een persoon uit het **andere** team moet het algoritme steeds uitvoeren.

Als ergens staat dat een teken op een willekeurige plek staat wordt er daadwerkelijk een **willekeurige** plek gekozen met een 9-zijdige dobbelsteen (bijv. <http://dice.virtuworld.net/>).

	Team 1	Team 2	Team 3
Team 1			
Team 2			
Team 3			