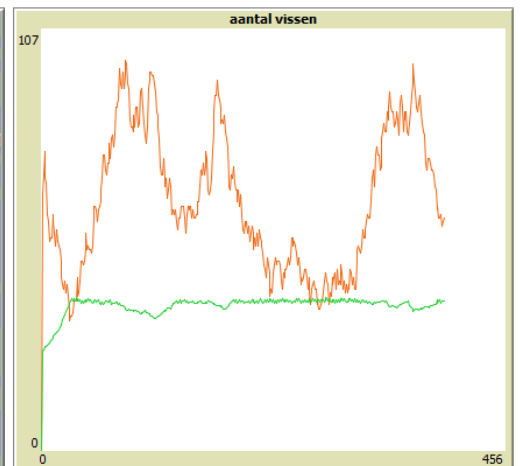
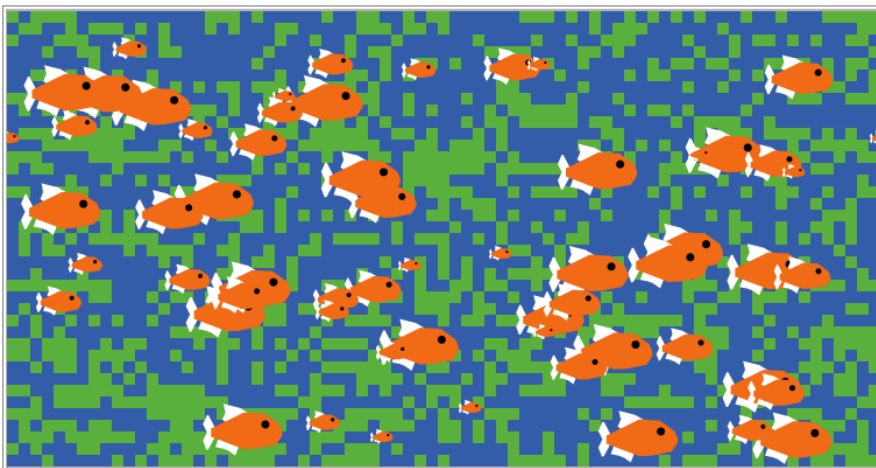


Agent-based modeling



Domein R: Computational Science

INHOUD

H1	modellen	4
1.1	Inleiding	4
1.2	Groepsgedrag	4
1.3	Tijd en iteraties	6
1.4	Het doel van modelleren	9
1.5	Onderzoek doen	12
1.6	De volledige modelleercyclus	15
H2	modellen maken	17
2.1	Inleiding	17
2.2	Een wereld met patches	18
2.3	patches maken en aanpassen	19
2.4	Bewegende agents	23
2.5	Agents die reageren op hun omgeving	27
2.6	Turtles die op elkaar reageren	30
2.7	Globale variabelen en statistiek	33
2.8	Eindopdrachten	36

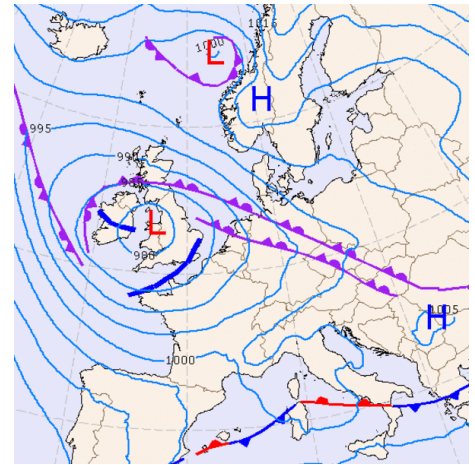
H1 MODELLEN

1.1 Inleiding

Een uitzending van het journaal eindigt standaard met het weerbericht. Hierin doet de weerman een voorspelling van het weer van de volgende dag en de dagen daarna met behulp van weerkaarten zoals in Figuur 1.1.

Dit soort voorspellingen wordt gedaan op basis van metingen van weerstations en ervaringen uit het verleden. Om tot een voorspelling te komen en daarmee weerkaarten voor in de toekomst te maken, wordt gebruik gemaakt van een computermodel. Dit soort wetenschap met behulp van modellen wordt *computational science* genoemd.

Je kunt een model gebruiken om een voorspelling te doen, maar ook om iets dat je hebt gezien te kunnen verklaren. Als je een vermoeden hebt hoe iets werkt, kun je dat met een computermodel gemakkelijk en eindeloos testen. Lukt het je om de werkelijkheid te simuleren? Dan is er een goede kans dat je hebt begrepen hoe die in elkaar zit!



FIGUUR 1.1

In dit hoofdstuk leer je een aantal belangrijke begrippen en aandachtspunten als je wilt gaan modelleren. Daarnaast oefen je met het nadenken over modellen en de werkelijke context die zij simuleren.

Opdracht 1: Modellen

1. In de inleiding wordt gesproken over een model van het weer. Noem minimaal drie voorbeelden anders dan een weermodel waarbij gebruik gemaakt wordt van computermodellen.
2. Het weermodel wordt gebruikt voor de weersvoorspelling. Het algemene doel van dit model is dan *voorspellen*. Bedenk minimaal drie andere algemene doelen van modelleren.
3. Wat betekent het woord *simuleren*?

Opdracht 2: Vuurmieren

Bekijk de video over samenwerkende vuurmieren.

1.2 Groepsgedrag

Vuurmieren (Figuur 1.2) zijn eenvoudige insecten, hoewel ze pijnlijk kunnen bijten. Toch lijken de vuurmieren in de video intelligent gedrag te vertonen. Hoe kan dat?

Dat gedrag is het gevolg van een aantal simpele **regels** waarmee de vuurmier genetisch is geprogrammeerd. De video laat zien dat een grote groep mieren samen een toren kunnen vormen door zich aan elkaar vast te klemmen. Individuele mieren worden niet geplet, omdat ze een simpele regel volgen: *als de druk te groot wordt, laat ik los*.



FIGUUR 1.2

Het simpele gedrag van individuele mieren leidt hier tot complex **groepsgedrag** dat je nooit zou zien als je de mieren afzonderlijk zou bestuderen. Dit groepsgedrag heet **emergent gedrag**. De losse mieren die dit gedrag veroorzaken, worden **agents** genoemd. De agents **reageren** op hun **omgeving** door één of meer simpele regels uit te voeren. Dat zorgt voor groepsgedrag dat je niet zomaar op basis van die simpele regels had kunnen voorspellen.

Je kunt het gedrag van een mierenkolonie simuleren met een computermodel. Hierbij kun je er voor kiezen om de kolonie als één geheel te beschouwen, maar om het groepsgedrag te verklaren, is het vaak beter om te kijken naar de leden van de kolonie: losse mieren dus. Die aanpak heet **agent-based modeling**.

Opdracht 3: Helden en Lafaards

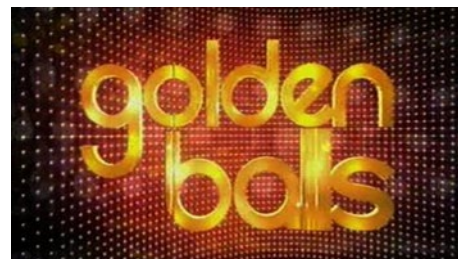


Bij dit spel zijn jouw klasgenoten en jij de agents. Je krijgt een paar spelregels van je leraar. We gaan kijken welk emergent gedrag dit voor de klas oplevert.

Opdracht 4: Delen of stelen



Aan het eind van de televisiespelshow *Golden Balls* (figuur 1.3; je kunt het ook op een computer spelen) krijgen de kandidaten de opdracht om de samen opgebouwde geldprijs (*jackpot*) te verdelen. Beide kandidaten moeten kiezen of ze het geld willen delen (*split*) of dat ze de jackpot van de ander willen stelen (*steal*). Maar... het bedrag dat je wint is mede afhankelijk van de keuze van de andere kandidaat.



FIGUUR 1.3

De spelregels zijn als volgt:

- Als beide spelers delen, krijgen ze allebei 50% van het geld
- Als beide spelers stelen, krijgen ze allebei niets (0%)
- Als één van de spelers deelt en de ander steelt, krijgt de stelende speler 100% (en de delende kandidaat niets)



4. Bekijk de video van *Golden Balls*.



5. Klik op het icoontje en speel zelf het spel.

Alice en Bob spelen het spel. De mogelijke uitkomsten van het spel zijn weergegeven in Figuur 1.4. Als Alice deelt, wint zij in het ene geval 50% van de jackpot en in het andere geval niets. Als Alice steelt, wint zij in het ene geval 100% van de jackpot en in het andere geval niets.

		Alice	
		A deelt	A steelt
Bob	B deelt	50%	100%
	B steelt	0%	0%

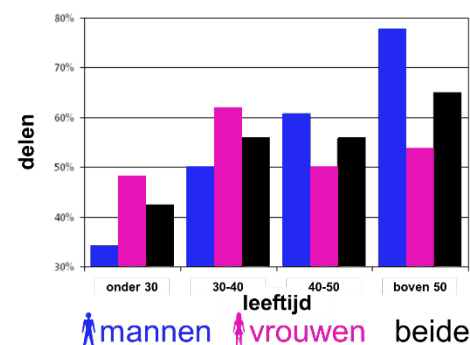
FIGUUR 1.4

Wiskundig kunnen we eenvoudig constateren dat stelen gunstiger is dan delen. Maar zo eenvoudig is het ook weer niet: als iedereen volgens die logica zou spelen, zou er nooit iemand winnen!

In Figuur 1.5 zie je hoe vaak deelnemers aan het spel in werkelijkheid kiezen voor delen. Wat zou jij doen?

- Bedenk minimaal drie onderzoeksvragen die je op basis van Figuur 1.5 zou kunnen beantwoorden.
- Wat zijn de antwoorden die horen bij de door jou bedachte vragen?
- Formuleer twee onderzoeksvragen bij dit spel die niet op basis van Figuur 1.5 kunnen worden beantwoord.
- Stel een hypothese (voorspelling) op bij de door jou bedachte onderzoeksvragen. Welke uitkomst verwacht je hierbij?

Figuur 1.4 brengt het spel schematisch in kaart, maar Figuur 1.5 gaat over menselijk gedrag. Op basis waarvan kiezen mensen? Waar wordt hun keuze door beïnvloed?



FIGUUR 1.5

- Bedenk drie factoren die de keuze van een kandidaat in de praktijk zouden kunnen beïnvloeden.

Statistieken heb je pas achteraf. Als je wilt voorspellen wat de uitkomst van een spel met bepaalde spelregels zal zijn, dan kun je dit doen met behulp van een model.

- Neem Figuur 1.4 over en pas de percentages met maximaal 5% aan zodanig dat er meer kandidaten zullen winnen, maar de spelshow niet meer geld kwijt is aan de winnaars. Licht je antwoord toe.

Opdracht 5: Weldoener



Wat zou er gebeuren als we allemaal een beetje weldoener waren en elke dag een euro aan iemand anders zouden geven? Bekijk de simulatie en beantwoord de daar gestelde vragen.

1.3 Tijd en iteraties

In de video van de vuurmieren hebben we gezien dat het gedrag van individuele mieren leidt tot collectief gedrag van de mierenkolonie. Dit emergente gedrag van de agents (losse mieren) was het gevolg van simpele gedragsregels, waarbij een mier reageert op zijn omgeving. Deze omgeving van de mieren, ofwel de kolonie, verandert voortdurend van vorm in de loop van de **tijd**. Mieren nemen daarom steeds opnieuw een beslissing over wat ze gaan doen volgens de gedragsregels waarmee ze zijn geprogrammeerd.

Het model *Weldoener* wordt ook in stappen uitgevoerd. Na elke stap (*tick*) kijkt elke mens (agent) of hij geld heeft en als dat zo is doneert hij een eenheid geld aan een ander. Ofwel: de regels van het model worden telkens opnieuw door alle agents uitgevoerd. Eén zo'n stap heet een **iteratie**. Na elke iteratie is het spel in een nieuwe **toestand** en zijn we een stapje of verder in de tijd. De iteratie is de tijdeenheid.

Een model ontwikkelt zich in de tijd, doordat het dezelfde regels keer op keer herhaalt. De uitvoer van een computermodel zal dan ook altijd bestaan uit meerdere iteraties. Na elke iteratie is er een nieuwe toestand. Bij het spel *Helden en Lafaards* heb je dat zelf ervaren.

Het blijkt dat veel processen op grotere schaal te modelleren zijn door ze te beschrijven op basis van het gedrag van kleinere eenheden. Het emergente gedrag is het gevolg van de gedragsregels van de agents: *Agent-based* modeling. Een prachtig voorbeeld hiervan is de voortdurend veranderende vorm van een zwerm spreeuwen. In Figuur 1.6 zie je de toestand van een zwerm spreeuwen op drie opeenvolgende momenten. Dit soort groepsgedrag zie je ook bij andere vogels, maar bijvoorbeeld ook bij scholen vissen.



FIGUUR 1.6

Opdracht 6: Een zwerm spreeuwen



Bekijk de video van een zwerm spreeuwen.

We willen een model maken van de vorm van een zwerm spreeuwen door het gedrag van individuele spreeuwen te beschrijven. De spreeuwen laten zich beïnvloeden door omgevingsfactoren.

12. Noem minimaal vijf factoren die het gedrag van individuele spreeuwen zouden kunnen beïnvloeden.
13. Welk nut zou een model van een zwerm spreeuwen kunnen hebben?
14. Wat zou voor dit model een geschikte tijdeenheid kunnen zijn? Wat is één iteratie?
15. Wonderbaarlijk genoeg komen de spreeuwen nooit met elkaar in botsing. Bedenk een toepassing waarbij de kennis van de achterliggende mechanismes van dit fenomeen bruikbaar kunnen zijn.

Opdracht 7: Bosbrand



Bekijk de video van een grote bosbrand. We willen een model maken van de verspreiding van zo'n brand.

16. Wat zijn de agents in zo'n model?
17. Zou *vuur* een agent kunnen zijn in dit model, denk je? Waarom wel / niet?
18. Hoe reageren de agents op hun omgeving? Van welke factoren is hun gedrag afhankelijk?
19. Welk doel zou een model van de verspreiding van een bosbrand kunnen dienen?

Opdracht 8: Iteraties



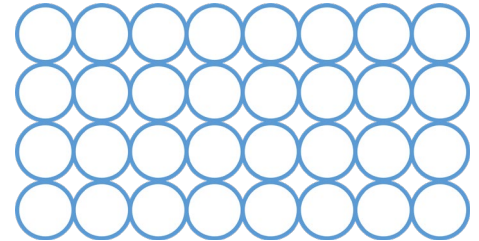
Je onderzoekt het emergente gedrag van een groep agents. Bij deze opdracht hoort een werkblad.

Opdracht 9: Laterale inhibitie



In de vorige opdracht heb je beredeneerd hoe agents reageren op hun omgeving op basis van slechts een paar regels. Dit soort regels komt vaker voor in de natuur. Ze kennen hele praktische toepassingen.

Het licht dat je via je ogen ontvangt, wordt opgevangen op je netvlies. Het netvlies bestaat uit een hele reeks piepkleine lichtgevoelige cellen (staafjes en kegeltjes) die het licht omzet in een elektrisch signaal dat naar je hersenen gaat. Deze cellen zijn schematisch getekend in Figuur 1.7. Hiermee maken we een vereenvoudigd model van het netvlies.



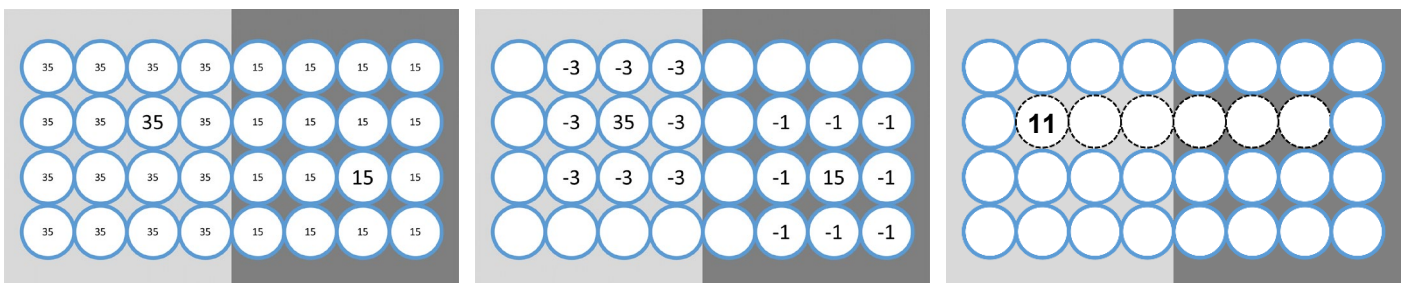
FIGUUR 1.7

Voor de cellen van het netvlies gelden de volgende regels:

- Als er een feller (lichtere) tint op de cel valt, reageert deze door een hoger signaal te geven (A)
- Cellen onderdrukken hun burens. Hoe feller de tint op de cel, des te meer ze hun burens onderdrukken (B)

Een persoon kijkt naar een vlak met twee grijstinten. De lichtgevoelige cellen reageren op de lichtere tint door een signaal met een sterkte van 35 af te geven. De donkere tint zorgt voor een signaal van 15 (figuur A). Dit komt overeen met de eerste regel.

De cellen op het netvlies zijn met elkaar verbonden door een complex netwerk van zenuwen. Dit stelt ze in staat om het signaal van hun burens te onderdrukken. De cellen links ontvangen veel licht en onderdrukken hun burens door 3 van hun signaal af te trekken. De cellen rechts ontvangen weinig licht en zenden daarom zelf een lager signaal uit. Ook onderdrukken ze hun burens met een afname van slechts 1 (figuur B). Dit is conform de tweede regel.



FIGUUR 1.8

A

B

C

In figuur C is een rij van zes cellen gemarkeerd met een stippelijntje. De eerste cel heeft ten gevolge van de twee regels uiteindelijk een uitgangssignaal van 11.

- Toon dit aan met een berekening.
- Bereken de uitgangssignalen van de overige gemarkeerde cellen.

Het fenomeen dat cellen het signaal van hun burens onderdrukken, heet laterale inhibitie.

- Bedenk op basis van de uitkomst van de vorige vraag wat het nut is van laterale inhibitie. Figuur 9 geeft hierbij misschien nog een hint. Je ziet twee keer dezelfde twee grijze vlakken!



Het fotobewerkingsprogramma Photoshop kan dit principe gebruiken.

- Klik op het video-icoon om de demonstratievideo te bekijken.
- Heb je Photoshop tot je beschikking? Maak dan je eigen *custom filter*.



FIGUUR 1.9

Opdracht 10: Spinnenwebben

Spinnen weven hun web met spinrag of spinsel. Als eerste maken ze een brug van spinrag tussen twee punten. Afhankelijk van de omgeving, volgt een aantal extra grote lijnen die een basisstructuur vormen. Daarbinnen maakt de spin zijn web. Niet alle soorten doen dat op dezelfde manier. Wij bekijken ronde webben, zoals bijvoorbeeld kruisspinnen die maken (Figuur 1.10).



FIGUUR 1.10

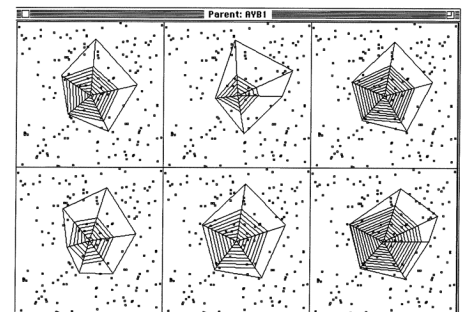
De vorm komt je vast bekend voor, maar waarom hebben spinnenwebben deze vorm en niet een andere vorm? En hoe weet een spin dat het effect zal hebben, wanneer hij zijn web op deze manier weeft? Het korte antwoord op die laatste vraag is dat de spin dat niet weet. Het antwoord op de vraag over de vorm kan gegeven worden met behulp van een model.

In zijn boek *Het toppunt van onwaarschijnlijkheid* (Engels: *Climbing Mount Improbable*) beschrijft de Britse evolutiebioloog Richard Dawkins een computermodel dat spinnen een web laat weven in meerdere iteraties. Het 2D-ontwerp van het spinnenweb ontwikkelt zich elke iteratie op basis van een beperkt aantal regels waarmee de spin genetisch is geprogrammeerd, net als de vuurmieren uit paragraaf 1.1.

Als een insect in een web vliegt, zit hij in de val, omdat hij blijft plakken aan het spinrag. Het model simuleert de plaats waar insecten door het oppervlak van het web vliegen en het aantal insecten dat langs komt. Een cruciale factor hierbij is willekeur: de plaats en het aantal zijn niet elke keer hetzelfde, maar *random*. Maar ook het ontwerp van de spinnenwebben is willekeurig. Spinnen erven een deel van hun bouwtechnieken van hun ouders, maar bij de voortplanting ontstaan kleine variaties of mutaties, waardoor spinnen anders worden geprogrammeerd en hun eigen web er net anders uit kan gaan zien dan dat van hun ouders.


25. Leg uit dat het afhangt van de vorm van het web of een spin zich zal kunnen voortplanten.
26. Gebruik de term *random* of *willekeur* om uit te leggen dat een spin met een goed ontworpen web zich soms toch niet zal overleven om zich voort te planten.

In Figuur 1.11 zie je spinnenwebben uit het computermodel. Linksboven is het web van een ouder. De overige vijf webben zijn van kinderen van deze ouder. Ze zijn verschillend, want door de toeval zijn er kleine aanpassingen in de manier waarop een spin genetisch geprogrammeerd is om het web te weven. De stippen in Figuur 1.11 zijn de insecten die in het model in de buurt van het web komen.

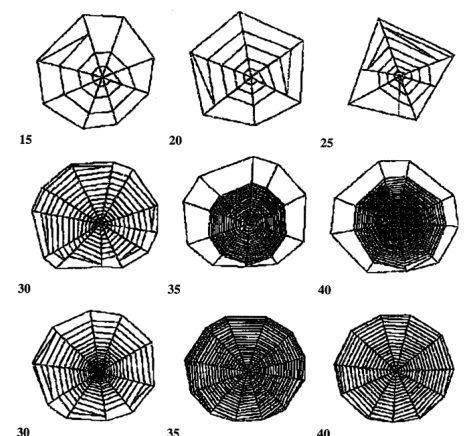


FIGUUR 1.11

27. Hoe zie je aan de stippen dat deze *random* gemodelleerd zijn?
28. Wat is de betekenis van één iteratie in dit model?
29. In het model wordt de ontwikkeling van een populatie spinnen gemodelleerd. Leg uit waarom het noodzakelijk is om met een populatie te werken. Waarom is één spin die zich in het model voortplant niet genoeg?

 In een interactief college in de jaren 90 legt Richard Dawkins uit hoe het model van de ontwikkelingen van spinnenwebben de theorie van natuurlijke selectie (evolutie) ondersteunt.

30. Bekijk de eerste zesentwintig minuten van de video.
31. Hoe bepaalt het computermodel het succes van het ontwerp van een spinnenweb? Dawkins noemt twee factoren.
32. Figuur 1.12 toont de ontwikkeling van het computerweb met stapjes van vijf generaties. Hoe kunnen de modelmakers vaststellen dat hun model een realistische simulatie van de werkelijkheid is?
33. Leg uit dat dit soort onderzoek moeilijk door middel van waarneming uit te voeren is.



FIGUUR 1.12

1.4 Het doel van modelleren

In de wetenschap, maar ook bij bedrijven en in de sport wordt steeds meer gebruik gemaakt van modellen. Waarom doen we dat? Het verhaal van de spinnenwebben in de vorige opdracht geeft hierop al meerdere antwoorden:

- het model geeft een manier om de werkelijkheid te **beschrijven**
- hiermee kunnen we de ontwikkeling van spinnenwebben **uitleggen** of **verklaren**
- in de video wordt het model gebruikt voor een college. Dit heeft als doel **onderwijs** of **educatie**
- met het model kun je eenvoudig nieuwe situaties proberen: je kunt **experimenteren**
- op basis van de resultaten van een model kun je gaan **discussiëren** en **voorspellen**

In het model van de spinnenwebben was sprake van **toeval** ofwel **random** gedrag. Dit zijn belangrijke begrippen als je gaat modelleren, omdat veel fenomenen die we kunnen modelleren die willekeur bevatten. Ze gaan nu eenmaal niet altijd precies op dezelfde manier. Dat noemen we **niet-deterministisch**. De willekeur in het model zit ten eerste in de insecten die in willekeurige hoeveelheden op willekeurige plaatsen langs en in het spinnenweb komen. Ten tweede zit er willekeur in de manier waarop een nieuwe generatie spinnen genetisch wordt geprogrammeerd om een web te weven. Een spin erft veel van die regels van zijn voorouders, maar door mutaties worden die regels een klein beetje random gewijzigd. Op die manier kan worden verklaard dat de vorm van de spinnenwebben in de loop van de tijd verandert.

Opdracht 11: Mensenstroom



Bekijk de video waarin wordt gemodelleerd hoe een groep mensen zich gedraagt wanneer een groot stadion in Sint-Petersburg (Rusland) moet worden geëvacueerd. De getoonde simulaties zijn gemaakt, nog voordat het stadion (figuur 1.13) in 2017 werd geopend.

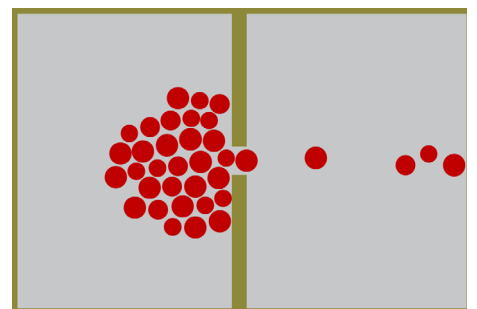


FIGUUR 1.13

34. In de theorie wordt een aantal mogelijke doelen van modelleren beschreven. Welke doelen zijn volgens jou van toepassing op deze situatie. Geef bij elk gekozen doel een toelichting.
35. In de video worden meerdere evacuatieplannen gesimuleerd. Wat is hiervoor de reden?
36. Wat zijn de agents in dit model?
37. Noem vijf factoren die het gedrag van de agents (mede) zullen bepalen.

Een voetbalstadion voor bijna 70 duizend toeschouwers is een hele ingewikkelde omgeving om te modelleren. In modellen wordt een situatie vaak vereenvoudigd door de omgeving te versimpelen en het aantal gedragsregels te beperken.

In figuur 1.14 zie je een beeld van zo'n versimpeld, abstract model van een ruimte. Het model simuleert een groep mensen die de ruimte (links) zo snel mogelijk wil verlaten, vanwege rookontwikkeling. Er is maar één opening (midden). Omdat iedereen snel weg wil, ontstaat een opstopping. Die is niet goed voor de totale ontruimingstijd.



FIGUUR 1.14

38. Bedenk minimaal drie omgevingsfactoren waar de totale ontruimingstijd van afhangt.
39. Bedenk drie onderzoeksvragen over de ontruimingstijd die je bij dit model zou kunnen stellen.
40. Voorspel wat de antwoorden op jouw eigen onderzoeksvragen zullen zijn.
41. Het model van figuur 1.14 is heel erg versimpeld. Mensen worden getekend als rode bolletjes. Hoe zie je dat rekening gehouden is met het feit dat mensen niet allemaal even groot zijn?
42. Bedenk twee manieren waarop de modelmakers willekeur of random gedrag in het model zouden kunnen hebben ingebouwd.



Hoe bepaalt de indeling van een lokaal de snelheid van de ontruiming? In de simulatie bij deze opgave kunnen we dit voor verschillende klasopstellingen vergelijken.

43. Bouw jullie eigen klasopstelling na en vergelijk de ontruimingstijd met de opstellingen in het model.

Opdracht 12: Kiezen I



Bekijk de video over het voorspellen van menselijk keuzegedrag bij verkiezingen. Hierbij wordt onder meer iets verteld over het gebruik van data en modellen tijdens de verkiezingsstrijd tussen Donald Trump en Hillary Clinton (figuur 1.15).

In de eerste minuten van de video wordt verteld dat er op basis van algoritmes wordt voorspelt wie binnenkort een nieuwe auto zal gaan kopen of wie fraude zal gaan plegen. Om dit te voorspellen gebruikt het model gegevens of data.



FIGUUR 1.15

44. Bedenk vijf persoonlijke gegevens die je zou kunnen gebruiken om te voorspellen of iemand binnenkort een nieuwe auto zal gaan kopen.
45. De uitkomst van de algoritmes zijn *niet-deterministisch*. Leg dit uit.
46. Welke reden geeft de onderzoekster (in de tweede minuut van de video) voor haar onderzoek naar de vraag wie er kans maakt om in de bijstand terecht te komen?
47. De campagnestrategie zegt in de derde minuut: *alles wordt met alles vergeleken en berekend*. Geef jouw mening over deze uitspraak.
48. In de vierde minuut wordt verteld op welke manier er profielen worden opgesteld van potentiële kiezers. Op welke manier worden deze gegevens gebruikt om de verkiezingsuitslag te beïnvloeden?
49. Wat is *microtargeting*?
50. Welke tactieken zijn door het Trump-campagneteam gebruikt om kiezers niet op de concurrent Clinton te laten stemmen?
51. Vind jij jezelf voorspelbaar? Waarom? Ben je makkelijk te manipuleren, denk je?

We willen met een sterk versimpeld model modelleren hoe een groep mensen elkaar kan beïnvloeden. Hiervoor maken we een ruimte met 100 mensen. De helft van de mensen krijgt de politieke voorkeur blauw, de andere helft de voorkeur geel, vergelijkbaar met *Democraat* of *Republikein*. De mensen worden over de ruimte verspreid. We gaan onderzoeken hoe mensen die naast elkaar staan elkaar kunnen beïnvloeden.

52. Deze verdeling van 50-50 is precies bepaald en dus niet willekeurig. Leg uit hoe je ondanks dit gegeven toch een random verdeling kunt krijgen.

De agents in dit model zijn de mensen. Ze krijgen twee eigenschappen die in grootte kunnen variëren:

- overredingskracht: hoe goed is iemand in staat om een ander met argumenten te overtuigen van het feit dat zijn politieke voorkeur de juiste is?
- vasthoudendheid: hoeveel is er voor nodig om iemand anders van mening te doen veranderen? Hoe overtuigd zijn ze van hun eigen gelijk?

Bij agent-based modeling vertonen agents hun gedrag op basis van een beperkt aantal simpele regels. De agents reageren hierbij op hun omgeving.

53. Wat is in dit model de omgeving?
54. Bedenk een gedragsregel: op basis waarvan kan de politieke voorkeur van een agent veranderen?
55. Wat zou in dit model een iteratie kunnen zijn?

Opdracht 13: Kiezen II



Op basis van een model onderzoek je hoe de politieke voorkeur van mensen kan worden beïnvloed door hun directe omgeving. Agents laten zich beïnvloeden door de overtuigingskracht van nabije agents. Als die overtuigingskracht groter is dan hun eigen vasthoudendheid, kunnen ze van politieke voorkeur wisselen.



Bij deze opgave hoort een werkblad.

-1	0	-2	1	0
-2	3	-3	5	3
-2	2	0	0	-1
-2	7	-3	1	-1
-1	0	0	1	0
-1	2	-3	2	4
-1	2	-2	1	0
-3	5	-5	3	-2

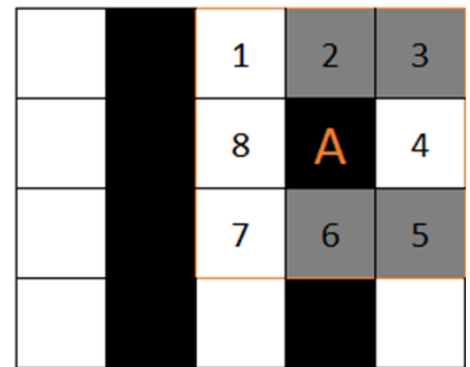
FIGUUR 1.16

Opdracht 14: Cellulaire automaten



Bij opdracht 8 (iteraties) en opdracht 13 (politieke voorkeur) heb je gewerkt met een abstracte wereld van agents in de vorm van vierkantjes. Agents veranderen van kleur op basis van een aantal eenvoudige gedragsregels, waarbij ze reageren op hun omgeving. Die omgeving bestaat uit acht omringende agents (zie Figuur 1.17).

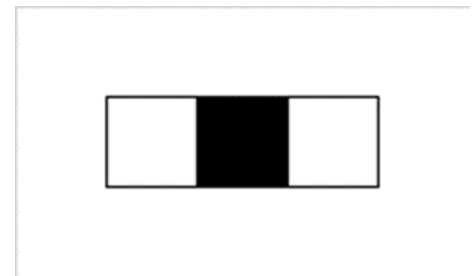
Als je de opdracht 8 en 12 hebt gemaakt heb je gemerkt dat het, zelfs met maar een paar gedragsregels en eigenschappen, al snel ingewikkeld wordt om te bepalen of voorspellen of een agent van kleur zal wisselen. Het emergente gedrag van alle agents samen – het totale patroon – is al helemaal niet te voorspellen. Daarom gebruiken we vanaf hoofdstuk 2 de computer om dit voor ons door te doen.



FIGUUR 1.17

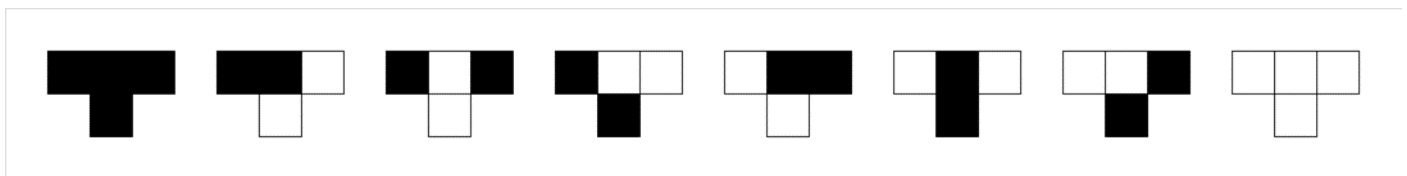
De modellen die we in opdracht 8 en 12 hebben bekeken zijn voorbeelden van cellulaire automaten. Een cellulaire automaat bestaat standaard uit een raster van cellen (agents) die zich in een beperkt aantal toestanden (meestal kleuren) kunnen bevinden. Door het toepassen van een aantal gedragsregels die betrekking hebben op de toestand van hun burens, kan de toestand van de cel veranderen.

In de meest elementaire vorm van een cellulaire automaat stellen we ons een cel in één dimensie voor met slechts twee burens zoals in Figuur 1.18. De cel is zelf zwart (waarde 1) en zijn burens zijn allebei wit (waarde 0).



FIGUUR 1.18

In Figuur 1.19 zie je dat er nu maximaal $2^3 = 8$ verschillende mogelijkheden zijn voor de drie cellen. We willen nu dat de cellen of agents gaan reageren op hun omgeving op basis van regels. Omdat er acht verschillende combinaties van de drie cellen mogelijk zijn, moeten we die voor alle acht gevallen beschrijven. Zie Figuur 1.19.



FIGUUR 1.19

In de figuur zie je in de bovenste rij de acht mogelijke combinaties van de cel met zijn twee burens. In de rij eronder staat hoe de cel in een volgende iteratie moet kleuren op basis van de huidige combinatie.

56. De acht combinaties zijn (van rechts naar links) heel bewust in een bepaalde volgorde getekend. Leg dit uit aan de hand van jouw kennis over binaire getallen.
57. Als je in de onderste rij voor een zwarte cel '1' invult en voor witte cel '0', dan ontstaat 10010110. Leg uit dat de situatie in Figuur 1.19 'regel 150' wordt genoemd.



Met behulp van een werkblad in Excel gaan we kijken welke patronen er ontstaan na een aantal iteraties.

58. Gebruik het Excel-werkblad om regel 150 voor een aantal opeenvolgende iteraties uit te werken, door de achtergrondkleur van de cellen waar nodig in zwart te veranderen.
59. Ga naar het tabblad met de naam *regel 57* en vul bovenaan (in rij drie de regels) aan door de juiste vakjes zwart op te vullen. Vul daarna vanaf regel 6 de vakjes aan voor een aantal iteraties.



Als je het patroon van vele iteraties wilt bekijken, kun je beter de computer laten tekenen.

60. Gebruik de link om regel 57 door de computer uit te laten voeren. Bekijk het patroon dat ontstaat.
61. Bekijk de regels 30, 45 en 225.
62. Gebruik het tabblad *eigen regel* in Excel om zelf een regel te ontwerpen en teken een paar iteraties bij je eigen regel. Laat jouw regel daarna online uitvoeren voor een groot aantal iteraties.

Opdracht 15: Game of life



Bekijk de video waarin een cellulaire automaat (zie vorige opgave) is gemaakt op basis van een aantal simpele regels die bekend staan onder de naam *Game of life*. Het is bedacht door de Britse wiskundige John Conway.

63. Gebruik internet om te onderzoeken op basis van welke simpele regels de agents zich gedragen.



Met de paar regels paar regels uit de vorige vraag kan verrassend complex emergent gedrag worden bereikt, waarbij vormen zoals die in Figuur 1.20 *tot leven* komen.

64. Open de simulatie bij deze opgave en probeer de mogelijkheden van de simulatie zelf te ontdekken.
65. Selecteer in het dropdown-menu achtereenvolgens *Glider*, *Exploder* en *10 Cell Row* en bekijk het resultaat.

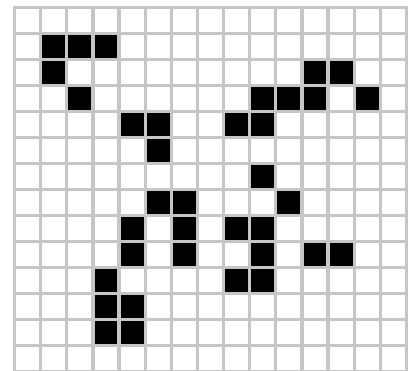


Er is veel onderzoek gedaan naar de *Game of Life* en vergelijkbare cellulaire automaten. Ook is er een aparte wiki-pagina aangemaakt waar veel informatie en voorbeelden te vinden zijn.

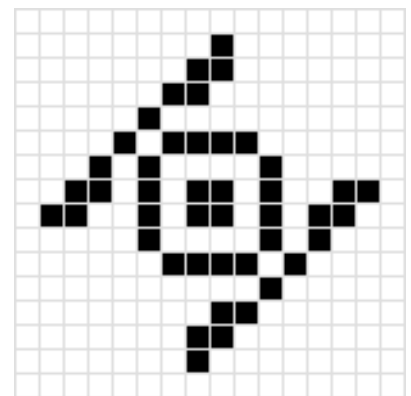
66. Klik op de link om naar de wiki-pagina te gaan. Wat is een *still life*? En wat is een *oscillator*?

In de simulatie kun je eigen vormen maken door op cellen te klikken.

67. Kies in het dropdown-menu voor *clear* en bouw vervolgens Figuur 1.20 na. Klik op *Start* om het resultaat te bekijken.
68. Doe hetzelfde voor figuur 21.



FIGUUR 1.20



FIGUUR 1.21

Opdracht 16: Verspreiding van HIV



Bij deze opdracht onderzoek je de verspreiding van het HIV-virus door onveilige seks. Jouw klasgenoten en jij zoeken contact en hebben 'gemeenschap'. De spelregels krijg je van je leraar.

1.5 Onderzoek doen

We hebben inmiddels nagedacht over een groot aantal modellen over de natuur (b.v. weer, bosbrand), dieren (b.v. vuurmieren, spreeuwen) en dierlijke gedragingen (netvlies, spinnenweb), menselijk gedrag (delen of stelen, vluchtgedrag, kiesgedrag) en een aantal abstracte modellen (cellulaire automaten, Game of Life). Ook hebben we een aantal situaties gesimuleerd (Helden en Lafaards, Weldoener). In alle gevallen was er sprake van gedrag van agents volgens een paar gedragsregels en konden we emergent gedrag van een groep agents waarnemen.

In de vorige paragraaf hebben we doelen van modellen benoemd. Maar hoe weet je nu dat het gedrag van een model klopt? Dat is best een moeilijke vraag. Het model dat de ontwikkeling van spinnenwebben modelleerde, voorspelt spinnenwebben die we ook in het echt in de natuur zien. Dat is een sterk argument in het voordeel van het model. Maar betekent dit ook dat de ontwikkeling van de vorm van een spinnenweb over miljoenen jaren net zo gaat als het model voorspelt?

Dit zijn vragen die je jezelf pas kunt stellen, nadat je een model hebt gemaakt en uitgevoerd, zodat je modelresultaten hebt. Vaak levert dit op dat je daarna je model verder wilt aanpassen of verfijnen. Voor je zover bent, moet je een aantal andere stappen doorlopen. De stappen om onderzoek te doen met behulp van een model worden op een rij gezet in de **modelleercyclus** met vijf fasen.

We beschrijven de modelleercyclus aan de hand van de *Mexican wave* (figuur 1.22). Volgens Wikipedia is dit *een gecontroleerde golfbeweging in een grote mensenmassa, die ontstaat wanneer groepen mensen beurtelings opstaan, vaak met de armen in de lucht, en weer gaan zitten.*



FIGUUR 1.22

FASE 1 definiëren

Een logische eerste stap is dat je beschrijft wat je wilt onderzoeken met het model. We hebben gekozen voor de *Mexican wave*, maar bij dit fenomeen kun je jezelf nog vele vragen stellen, zoals *hoe ontstaat een wave* of *hoe snel beweegt een Mexican wave door het stadion?* Perk je onderzoek in tot één concrete vraag: uitbreiden kan altijd nog!

Het is noodzakelijk om je af te vragen of het fenomeen dat je wilt onderzoeken wel past bij *agent-based modeling*. Dat is het geval wanneer er groep agents aan te wijzen is die individueel op elkaar en hun omgeving reageert. Het emergente gedrag van de groep is of lijkt hierbij afhankelijk te zijn van het gedrag van de agents en ontwikkelt zich in de tijd. Onze *Mexican wave* lijkt aan deze eisen te voldoen.

Wij kiezen voor de **onderzoeksvraag** *hoe snel beweegt een Mexican wave door het stadion?* Aan deze onderzoeksvraag kun je ook een **hypothese** (voorspelling) en een **doel** verbinden.

FASE 2 conceptualiseren

Een model simuleert meestal een extreme versimpeling van de werkelijkheid. In de opdracht over de *mensenstroom* (in een voetbalstadion bij een evacuatie) zien we individuele mensen als bolletjes van verschillende grootte die door de ruimte bewegen. Zo'n versimpeling, waarbij we kiezen wat wel en niet meedoet in het model, wordt **abstractie** genoemd. Je beslist nu: wat is nuttig voor het model en wat niet?

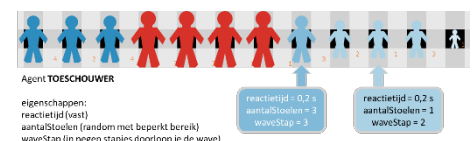
Door te versimpelen ontstaat een conceptueel model, dat we kunnen weergeven in de vorm van een tekst of een tekening. Het is een basale beschrijving van de echte wereld, waarbij je al rekening houdt met de modeltechnieken die je in fase 3 gaat gebruiken. Hierbij moet je vragen beantwoorden zoals:

- Wat zijn de agents in dit model? (Zijn ze zelfstandig?)
- Welke **eigenschappen** en welk gedrag hebben de agents en wat is daarvan van belang voor de onderzoeksvraag?
- Hoe **reageren** de agents op hun omgeving? Hoe worden ze beïnvloed?
- Wat is een **iteratie** in dit model? Wat gebeurt er in een iteratie en in welke volgorde?

In de opdracht over *kiezen* (tijdens verkiezingen) blijft er van mensen niet meer over dan een cel (vierkantje) die van kleur verandert. De enige eigenschappen in het model zijn *politieke voorkeur*, *overtuigingskracht* en *vasthoudendheid*. Blijkbaar vond de modelmaker eigenschappen als *geslacht* en *leeftijd* niet van belang of vond hij dat deze indirect terugkwamen in de andere eigenschappen. Dat wil niet zeggen dat deze eigenschappen in werkelijkheid niet van belang zijn! Maar in deze fase weet je dat niet: je bent immers onderzoek aan het doen. In het algemeen begin je simpel: uitbreiden kan altijd nog!

Bij ons model van de *Mexican wave* ligt de keuze voor agents voor de hand: individuele toeschouwers. We gaan er vanuit dat de agents reageren op andere agents. We kiezen de eigenschap *reactietijd*; deze geven we in eerste instantie een vaste waarde. Een mogelijke uitbreiding van ons model zou kunnen zijn dat we dit persoonsafhankelijk (random) maken.

Op wie reageren de agents nu precies? Op hun directe burenen? Of kijken ze twee of drie stoelen verderop of nog veel verder? We gaan ervan uit dat dit per agent varieert en nemen dit op in de eigenschap *aantalstoelen* die we random instellen. Verder versimpelen we het stadion tot één rij mensen op een vaste onderlinge afstand die er allemaal even lang over doen om te gaan staan en zitten.



FIGUUR 1.23

In de volgende paragraaf gaan we verder met de beschrijving van de modelleercyclus.

Opdracht 17: Mexican wave I

Bekijk de video van een *Mexican wave* in een voetbalstadion.

69. Definieer twee onderzoeksvragen (anders dan in § 1.4) met betrekking tot de *Mexican wave*.
70. Maak bij beide onderzoeksvragen een hypothese: wat verwacht je qua uitkomst?

In de vorige paragraaf is de volgende onderzoeksvraag gesteld:
Hoe snel beweegt een Mexican wave door het stadion?

Op basis van de video kunnen we een schatting maken van de uitkomst, zodat we in onze hypothese een getalswaarde kunnen noemen.

71. Bestudeer de video. Op basis van welke gegevens kun je een onderbouwde schatting doen van de snelheid waarmee de wave door het stadion beweegt?
72. Gebruik de gegevens uit de vorige opdracht om een hypothese te formuleren met behulp van een schatting op basis van de gegevens uit de vorige vraag.
73. In de theorie zijn in fase 2 (conceptualisatie) meerdere keuzes gemaakt die vallen onder abstractie. Benoem minimaal vijf van deze keuzes. Gebruik hierbij eventueel figuur 1.23.

Opdracht 18: Agressie

Bekijk de video over het gebruik van modellen door de politie. In Figuur 1.24 zie je een scherm uit één van de gebruikte modellen.

74. In de video wordt gesproken over *big data*. Wat wordt daarmee bedoeld?

Vanaf 2:20 in de video vertelt de medewerker dat de politie in staat is om een vechtpartij te voorspellen op basis van computermodellen.

75. Is het gebruikte model een voorbeeld van agent-based modeling? Leg uit waarom jij vindt dat dit wel of niet zo is.

De context *agressie van jongeren in het uitgaansleven* kan wel degelijk met agent-based modeling worden gemodelleerd. Om tot een conceptueel model te komen, beantwoorden we de volgende vragen.

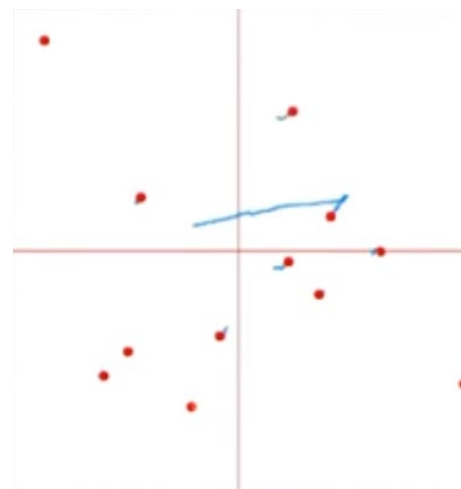
76. Wat zijn de agents in het model?
77. In Figuur 1.24 zie je rode bolletjes en blauwe strepen. Leg van beide uit wat hun betekenis is.
78. Het beeld van Figuur 1.24 is een abstract beeld. Wat wordt er bedoeld met *abstractie*?
79. Welke eigenschappen zijn voor het model van belang?
HINT: vanaf 3:50 worden een aantal factoren genoemd die door de politie worden gebruikt.
80. Leg van elk van de door jou genoemde factoren bij de vorige vraag uit welke invloed deze heeft op het gedrag van de agent.
81. Bedenk bij elk van de door jou genoemde factoren een onderzoeksvraag.

Vanaf 4:50 wordt gesproken over persoonsgegevens die in een (ander) model worden gestopt. Dit model kan gaan helpen bij de opsporing van mensen die een strafbaar feit hebben gepleegd.

82. Is dit model een voorbeeld van agent-based modeling? Waarom wel / niet?

Vanaf 6:00 wordt gesproken over het problemen die kunnen ontstaan als de modelmaker bepaalde vooroordelen heeft. Dat probleem zou zich ook kunnen voordoen in ons agressie-model.

83. Noem een voorbeeld van een eigenschap van een agent of van agent-gedrag dat op een vooroordeel gebaseerd zou zijn.



FIGUUR 1.24

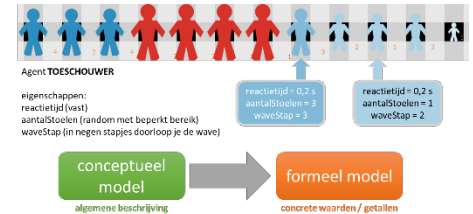
1.6 De volledige modelleercyclus

In de vorige paragraaf hebben we de eerste fasen van de modelleercyclus bekeken aan de hand van de casus *Mexican Wave* en gezien wat er nodig is om via definitie en vraagstelling tot een conceptueel model te komen. Welke stappen doe je daarna?

FASE 3 formaliseren

Modelleren doe je met de computer. Van ons conceptuele model van de *Mexican Wave* moeten we daarom naar een model waarin alles precies (eenduidig) is omschreven. Dit is de fase van systeemontwikkeling. Ons eindresultaat is een formeel model, zoals een wiskundige formules of een computerprogramma. Hoe je dat zelf doet, bekijken we in hoofdstuk 2.

Om te kunnen rekenen heeft een computer getallen en regels nodig. Aan de agent-eigenschappen *reactietijd* en *aantalstoelen* moeten we dus waarden gaan toekennen. De *reactietijd* geven we een vaste waarde van 0,2 s (200 milliseconde). Voor de eigenschap *aantalstoelen* laten we de computer willekeurig kiezen tussen 1 en 5. Bij het maken van de wave-beweging kiezen we ervoor om de beweging in negen stapjes (*waveStap*) op te delen die allemaal even lang duren (namelijk: 1 iteratie). Zie figuur 1.25.



FIGUUR 1.25

Bij het kiezen van deze waarden kun je gebruik maken van eigen ervaringen, maar ook van bronnen. In ons geval is de reactietijd op basis van een bron. Voor het aantal stoelen geldt: we weten het niet, maar we kunnen het later aanpassen als we zien welke invloed dit heeft op de uitkomst. Iets vergelijkbaars geldt voor de beweging. Het is een versimpeling waarvan we moeten ontdekken of hij gerechtvaardigd is.

Daarom controleer je na het bouwen de werking van het model. Dat doe je door **verificatie** en **validatie**. Bij verificatie check je of het model wel voldoet aan de omschrijving zoals jij die had bedacht: klopt de uitkomst van het model met wat je erin gestopt hebt? Gedragen de agents zich zoals jij het had bedoeld? Klopt het met ons eigen conceptuele model?

Dat het model een resultaat geeft, betekent nog niet dat die uitkomst klopt met de werkelijkheid. Daarom moeten we het model valideren. Dit doe je door de modelresultaten te vergelijken met wat je weet of met wat bronnen er over vertellen. Dat vergelijken doe je op twee niveaus.

Als je controleert of de agents zich gedragen zoals verwacht, doe je aan **microvalidatie**. Bij **macrovalidatie** kijk je of het hele systeem het gedrag vertoont dat je verwacht. In ons geval zou je bijvoorbeeld kunnen kijken naar een video van een *Mexican wave*. Hoe breed is het stadion? Hoe snel gaat de wave door het stadion? Lijkt de door jou gemodelleerde wave op die in de video? Soms is er al iemand die hier onderzoek naar heeft gedaan. In dat geval kun je die resultaten vergelijken met jouw model.

Voor zowel de verificatie als de validatie is het belangrijk dat je het model meer dan één keer uitvoert, bijvoorbeeld omdat we toeval (random waarden) in de eigenschappen hebben gestopt. Onthoud de regel: *one run is no run!*

FASE 4 experimenteren

Nadat het model geverifieerd en gevalideerd is, kunnen we experimenteren. Experimenteren is dus niet het voor het eerst uitvoeren van je model! Het is verder onderzoek doen als je kritisch naar de werking van je eigen model hebt gekeken en hebt beslist dat het model in eerste instantie *af* is. Natuurlijk weet je daarbij niet altijd 100% zeker dat je model voldoet, ook al heb je de verificatie en validatie zorgvuldig gedaan. Maar als het model realistische uitkomsten geeft voor bekende situaties, is dat een goede basis om er iets nieuws mee te voorspellen.

FASE 5 analyseren

De uitkomsten van jouw model zijn in de vorm van bijvoorbeeld getallen, grafieken, plaatjes of animaties. Jij moet daar zelf een betekenis aangeven. Die uitleg van de resultaten van het model heet de interpretatie. Jouw analyse levert het uiteindelijke resultaat van jouw onderzoek en een antwoord op de onderzoeksvraag.

De hele cyclus en reflectie

In Figuur 1.26 staan alle vijf fasen uit de modelleercyclus nog eens op een rij. Ze zijn bewust in een cirkel (cyclisch) weergegeven. In het kort kunnen we de modelleercyclus als volgt samenvatten:

- Definiëren: het probleem wordt beschreven in context; vragen worden geformuleerd.
- Conceptualiseren: de context wordt vertaald naar een abstract model door agents aan te wijzen en te bepalen welke factoren uit de echte wereld worden meegenomen en welke niet.
- Formaliseren: het conceptuele model wordt omgezet (geprogrammeerd) naar een computermodel (formeel model) dat wordt geverifieerd en gevalideerd.
- Experimenteren: het model wordt gebruikt om de onderzoeksvraag te beantwoorden.
- Analyseren: de modelresultaten worden geïnterpreteerd en vertaald naar conclusies in de echte wereld.



FIGUUR 1.26

De analyse van de uitkomsten van een experiment geeft vaak aanleiding tot nieuwe onderzoeksvragen. Voorbeeld: *wat is de invloed van een onder- of bovenliggende rij toeschouwers?* Zo'n vraag zorgt ervoor dat het conceptuele model en het formele model weer moeten worden aangepast met bijvoorbeeld nieuwe eigenschappen van de agents. Daarna moet opnieuw verificatie en validatie plaatsvinden, voordat opnieuw kan worden geëxperimenteerd.

Je doet er goed aan om simpel te beginnen qua onderzoeksvraag en model. Nadat je de cyclus hebt doorlopen, kun je langzaam toewerken naar moeilijkere vragen die alleen met een ingewikkelder model te beantwoorden zijn.

Cruciaal tijdens de hele cyclus is de **reflectie**. In elke fase moet je jezelf vragen stellen zoals: *Gaat het goed? Klopt het nog? Waar kan het beter?* Die reflectie zit deels in de verificatie en validatie, maar dat doe je pas als je een computermodel hebt gemaakt. *Zijn de agents achteraf gezien goed gekozen? Is er te veel versimpeld waardoor belangrijke eigenschappen van agents buiten beeld zijn gebleven?* Reflectie levert nieuwe vragen en wensen, maar ook nieuwe kennis.

Opdracht 19 Mexican wave II

Een modelmaker heeft een formeel model gemaakt voor een *Mexican wave*. Daarbij heeft hij de omschrijvingen uit de vorige twee paragrafen gevolgd en een aantal toevoegingen gedaan, zodat de onderzoeksvraag *hoe snel beweegt een Mexican wave door het stadion?* Met het model kan worden beantwoord.

84. Bekijk het model bij deze opgave en voer de daar getoonde opdrachten uit.

85. Stel dat je bij de verificatie op iets stuit dat niet overeenkomt met het conceptuele model dat je hebt opgesteld. Heeft het dan nog zin om te valideren? Waarom wel / niet?

Opdracht 20 Mexican wave III

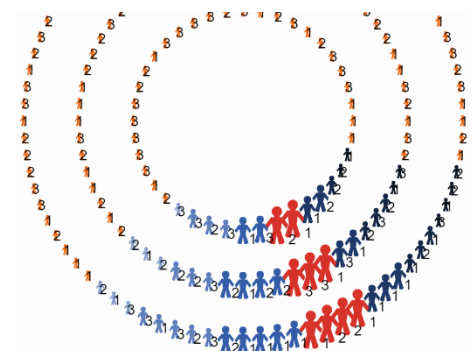
Een modelmaker heeft een formeel model gemaakt voor een *Mexican wave*. Daarbij heeft zij een aantal aanpassingen gedaan ten opzichte van de versie uit de vorige opdracht, zodat een rond stadion ontstaat.

86. Bekijk het model bij deze opgave en voer de daar getoonde opdrachten uit.

Onze onderzoeksvraag bij deze casus luidde:

Hoe snel beweegt een Mexican wave door het stadion?

87. Wat vind je na het maken van deze opdracht van de formulering van deze vraag?



FIGUUR 1.27

H2 MODELLEN MAKEN

2.1 Inleiding

In het vorige hoofdstuk heb je kennis gemaakt met modelleren, en in het bijzonder het modelleren volgens het principe van **agent-based modeling**. Er zijn verschillende programma's beschikbaar waarin je met modellen kunt werken. In dit hoofdstuk leer je werken met een van deze programma's genaamd **NetLogo**. NetLogo kun je gratis downloaden via <https://ccl.northwestern.edu/netlogo>.

Je leert in dit hoofdstuk werken met NetLogo door het maken van opdrachten. Stap voor stap leer je daarbij om zelf een model te maken, zodat je straks in hoofdstuk 3 zelfstandig een model kunt ontwerpen en maken om daarmee onderzoek te doen. Voor de handelingen in NetLogo kun je de opdrachttekst volgen, maar bij veel opgaven staat ook een video-icoon. Wie daar op klikt, krijgt een instructievideo waarin de stappen uit de betreffende opgave worden getoond.

Bij NetLogo wordt een groot aantal modellen meegeleverd die je meteen kunt proberen. De modellen zijn te vinden in de bibliotheek van NetLogo: de *models library*. Als rode draad door dit hoofdstuk loopt het model *Aquarium* dat we steeds verder zullen uitbreiden en waardoor je de belangrijkste stappen leert om in hoofdstuk 3 zelf een model te maken. We beginnen met een bekend model uit Hoofdstuk 1: *Bosbrand*.

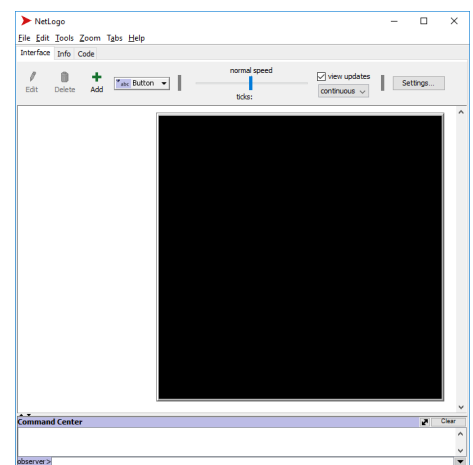
Opdracht 1: Bosbrand I

In deze opdracht leer je hoe je een model kunt vinden in de modelbibliotheek en hoe je hem kunt openen en uitvoeren.

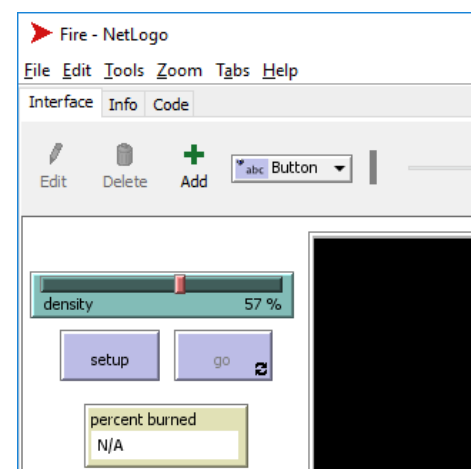
1. Open NetLogo. Als het goed is zie je het venster uit figuur 2.1.
2. Kies in het menu voor *File* → *Models Library*.
3. Onder de voorbeeldmodellen (*Sample Models*) kies je nu voor *Earth Science* → *Fire*. Rechts in het scherm verschijnt een korte omschrijving van het model. Klik op *Open*. Aan het venster van figuur 2.1 zijn linksboven de onderdelen uit figuur 2.2 toegevoegd.

Voor je een model kunt uitvoeren, heb je een beginsituatie nodig: er moet een wereld worden gemaakt met agents, vaak op basis van parameters (zoals *density*). Die wereld krijg je via *Setup*. Daarna is het mogelijk om op *Go* te klikken om het model uit te voeren.

4. Klik op *Setup* om groene bomen in het simulatiescherm te laten verschijnen. Klik daarna op *Go* om het model te starten.
5. Waar begint het vuur? Waar dooft het?
6. Klik nu nog een keer op *Setup*. Wat valt je op? Gebruik in je antwoord de term *random* uit hoofdstuk 1.
7. Klik op *Go* om het model opnieuw te laten lopen. Verklaar waarom je nu een andere uitkomst krijgt dan de eerste keer.
8. Boven de *Setup*-knop zit een zogenaamde **slider** met de naam *density* (figuur 2.2). Wat is de betekenis binnen dit model?
9. Verander een aantal keren de waarde van *density* en voer na elke aanpassing het model opnieuw uit.
10. Noem twee redenen waarom je het model vaker moet *runnen* als je onderzoek wilt doen.
11. Formuleer een onderzoeksvraag die je met dit model zou kunnen beantwoorden en stel een hypothese (voorspelling) op.
12. Onderzoek jouw vraag met het model. Wat is de uitkomst? Klopt die uitkomst met jouw hypothese?
13. Vlak onder het menu staan drie tabbladen, waarvan *Interface* nu actief is. Klik op het tabblad *Info* voor meer achtergrondinformatie. Lees de eerste twee blokken *WHAT IS IT?* en *HOW IT WORKS*. In dit tweede blok staat een element waarmee je het model kunt uitbreiden. Welke element is dat?



FIGUUR 2.1



FIGUUR 2.2

2.2 Een wereld met patches

Op het moment dat je in opdracht 1 op *Setup* klikte, veranderde het zwarte vlak in figuur 2.1 in een wereld met bomen. Die wereld is opgebouwd uit kleine tegeltjes. Deze heten in NetLogo **patches**.

In figuur 2.3 zie je de **Model Settings** die je kunt openen door bovenaan op *Settings...* te klikken. Hier kun je instellen hoe groot de modelwereld is door aan te geven hoeveel patches er horizontaal (*max-pxcor*) en verticaal (*max-pycor*) mogen zijn en hoe groot ze op het scherm moeten worden getoond (*Patch size*).

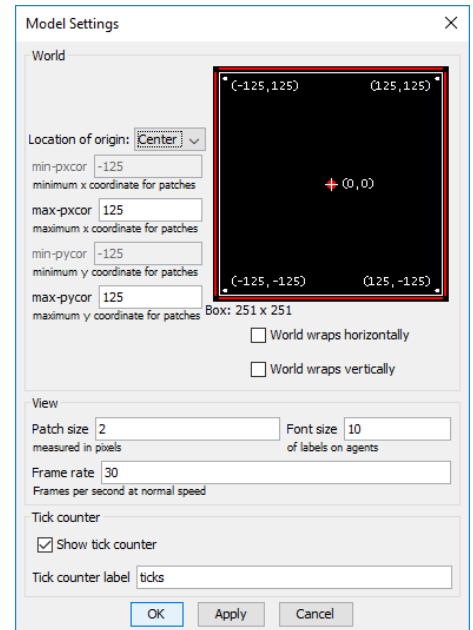
Hiermee heb je niet meteen het aantal patches. Je maakt hiermee een assenstelsel. Standaard wordt de oorsprong (*Location of origin*) van dit assenstelsel in het midden van de wereld (*center*) gezet. De opgegeven x- en y-coördinaten bestaan dan zowel in de plus- als de minrichting en voor elke combinatie bestaat er een patch.

Een **patch** kun je zien als een object. Elke patch heeft standaard de volgende **attributen** of eigenschappen:

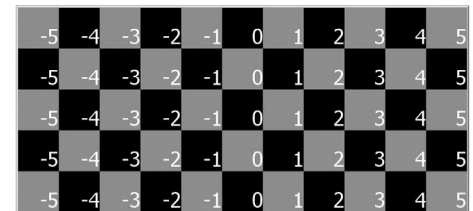
- *pxcor* de x-coördinaat
- *pycor* de y-coördinaat
- *pcolor* de kleur
- *plabel* een label die informatie over de patch kan bevatten
- *plabel-color* de kleur van de tekst van het label

Opmerking: je kunt zelf nieuwe eigenschappen (attributen) toevoegen aan een patch. Verderop in dit hoofdstuk leer je hoe dat moet.

In figuur 2.4 zie je de opbouw van een modelwereld met patches. Als *plabel* is gekozen voor de waarde van *pxcor*. De waarde *max-pxcor* is ingesteld op 5, zodat er patches zijn van -5 tot en met +5 (inclusief 0!).



FIGUUR 2.3



FIGUUR 2.4

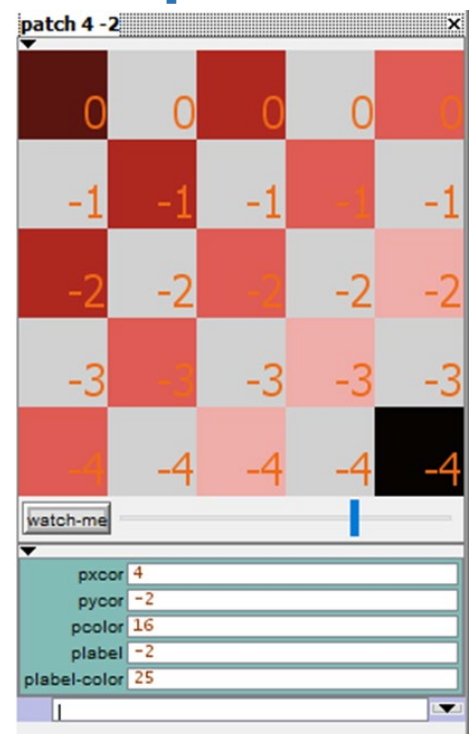
Opdracht 2: Modelwereld bekijken en aanpassen



14. Open het bestand *H2opg2.nlogo* en klik op *Setup*.
15. Hoeveel patches zijn er? Wat is dus de ingestelde waarde van *max-pxcor* en *max-pycor*?
16. Welke patch-eigenschap laat *plabel* zien?
17. Klik met de rechtermuisknop op de modelwereld en kies *Edit...* om bij de *Model Settings* (figuur 2.3) te komen. Wat is het antwoord op vraag 16 en hoe laat *Model Settings* dit antwoord zien?
18. Pas de wereld aan, zodat deze bestaat uit 15×9 patches.
19. De modelwereld wordt nu best groot weergegeven. Verander de weergave van de patches naar *Patch size* is 40. Kun jij op jouw scherm nu de hele modelwereld zien? Zo niet: pas de grootte van de patches dan aan zodat dit wel het geval is.
20. Klik op *Patroon*. Welke patch-eigenschap laat *plabel* nu zien?

De eigenschappen van een patch kunnen overzichtelijk worden weergegeven door hem te inspecteren (figuur 2.5).

21. Zoek de patch met de coördinaten $x = -3$ en $y = 3$. Klik met je rechtermuisknop op de patch en kies *inspect patch -3 3*. Wat is de kleurcode van deze patch?
22. De programmeur heeft een extra eigenschap aan de patches toegevoegd. Welke is dat?



FIGUUR 2.5

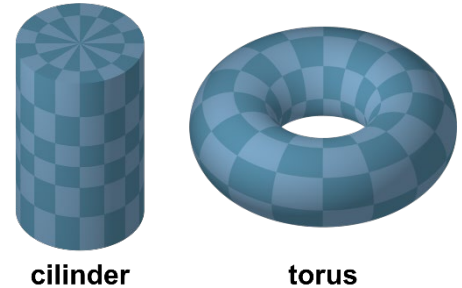
Opdracht 3: Bosbrand II | world wrap

23. Open het model *Fire* via *File* → *Models Library*; kies *Sample Models* → *Earth Science* → *Fire*.
Je kunt er ook voor kiezen om *fire* in te typen in het zoekvak (linksonder) van de *Models Library*.
24. Uit hoeveel patches bestaat deze modelwereld?
25. Verdubbel het aantal patches door *max-pxcor* twee keer zo groot te kiezen en maak *Patch size* 1.
26. Zet de parameter *density* op 61%. Voer vervolgens het model enkele keren uit. Is er veel variatie in de uiteindelijke percentage verbrande bomen (*percent burned*)?

Misschien ken je het begrip **wraparound** van games. Hiermee wordt aangegeven op welke manier de wereld begrensd is of juist verbonden is.

In de *Model Settings* staan twee settings om de **wraparound** in te stellen. Als je kiest voor *World wraps horizontally*, dan verbind je de linker- en rechterkant van de wereld met elkaar. De wereld wordt dan *opgerold*, zodat je een cilinder krijgt. Zie figuur 2.6.

Als je de patches niet alleen horizontaal maar ook verticaal met elkaar verbindt, ontstaat een figuur in de vorm van een donut: een torus.



FIGUUR 2.6

27. Voorspel wat er gebeurt als je het model *Fire* uitvoert in een cilindervormige wereld.
28. Controleer je voorspelling door in de *Model Settings* de eigenschap *World wraps horizontally* aan te vinken.
29. Verander de modelwereld in een torus en bekijk het resultaat als je het model uitvoert.

2.3 patches maken en aanpassen

In de vorige paragraaf leek het misschien alsof patches zomaar ontstaan en eigenschappen (attributen) krijgen, maar niets is minder waar. De buttons waarop je klikt (b.v. *Setup*) roepen programmacode aan die is geschreven in NetLogo. Je vindt de programmacode door op het tabblad *Code* te klikken, vlak onder het menu. Je ziet dan coderegels zoals het voorbeeld in figuur 2.7.

In dit voorbeeld is *setup* een **procedure** zoals je die waarschijnlijk kent van de programmeertalen. Alle code tussen *to setup* en *end* hoort bij de procedure en wordt dus uitgevoerd als *setup* – dat is de naam van de procedure – wordt aangeroepen (b.v. met een button).

Je kunt zelf kiezen welke naam je aan een procedure geeft, maar het is gebruikelijk om *setup* te gebruiken als procedurenaam om de begininstellingen van het model klaar te zetten. Die eerste stap heet **initialisatie** en begint bijna altijd met *clear-all*. Hiermee wordt alles (agents, variabelen, instellingen) dat eerder is gebruikt gewist.

Vervolgens kun je de patches opdrachten geven (*vragen*) met *ask patches*. Tussen de *[]* kun je één of meerdere opdrachten kwijt. In het voorbeeld van figuur 2.7 vragen we alle patches om een bepaalde kleur aan te nemen (*set pcolor 8*; 8 is een kleurcode die hoort bij een grijs tint) en om als label (tekst) de waarde van de y-coördinaat van de patch te tonen (*set plabel pycor*).

Het commando *set* is de algemene manier om een **attribuut** (of eigenschap) van een patch of een andere variabele te veranderen. Dat hoeft niet altijd voor alle patches tegelijk. In figuur 2.7 zie je een tweede opdracht waarbij alleen aan patches met een y-coördinaat groter of gelijk aan 0 wordt gevraagd om een **blauwe** kleur aan te nemen met kleurcode 106.

Hoe weet je nu welke kleur bij welke kleurcode hoort? Kies in het menu voor *Tools* → *Color Swatches* om een overzicht te krijgen van de codes en namen (zoals **lime** en **magenta**) die je mag gebruiken.

```
to setup
  clear-all
  ask patches [
    set pcolor 8
    set plabel pycor
  ]

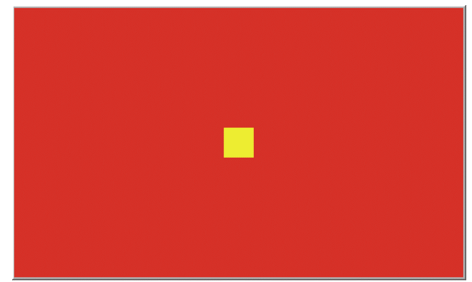
  ask patches with [pycor >= 0] [
    set pcolor 106
  ]
end
```

FIGUUR 2.7

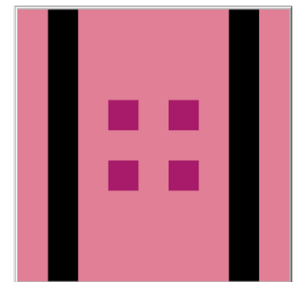
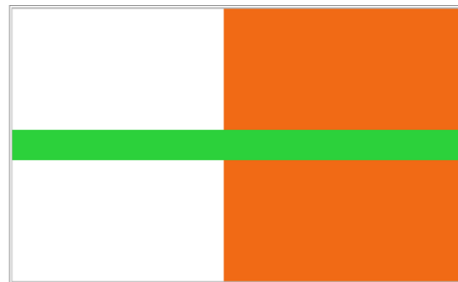
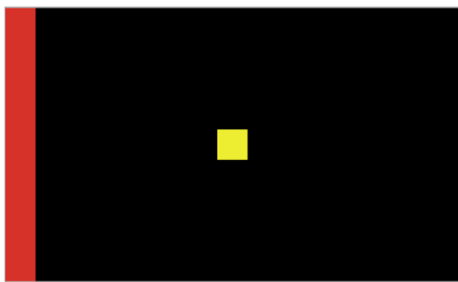
Opdracht 4: Zelf patches aanpassen



30. Open het bestand *H2opg4.nlogo* en klik op *Setup*.
31. Open het tabblad *Code*. Op welke manier kun je één specifieke patch vragen om een opdracht uit te voeren?
32. Pas de procedure *setup* aan, zodat de modelwereld er (na het klikken op *Setup*) uitziet zoals in figuur 2.8.
33. In figuur 2.9 staan nog drie afbeeldingen van de modelwereld. Gebruik de theorie (van § 2.3), inclusief *Color Swatches*, om ze na te maken. Merk op dat je voor de rechter afbeelding ook de afmetingen van de modelwereld moet aanpassen!



FIGUUR 2.8



FIGUUR 2.9

Opdracht 5: Logische operatoren

Bij de vorige opdracht heb je instructies gegeven aan een groep patches die je selecteerde op basis van één bepaalde eis (zoals *patches met x-coördinaat gelijk aan 0*). We kunnen eisen combineren door gebruik te maken van **logische operatoren**, zoals **and**, **or** en **not** (!=). Waarschijnlijk heb je die al eens gebruikt tijdens de programmeerlessen.

Door slim eisen te combineren, kun je met minder regels code hetzelfde bereiken. Hieronder staat een aantal voorbeelden om een deel van de patches te selecteren:

- i. `ask patches with [pxcor = 0 or pycor = 0]`
- ii. `ask patches with [pxcor = 0 and pycor = 0]`
- iii. `ask patches with [pxcor != 0 or pycor = 0]`
- iv. `ask patches with [pxcor = 0 and pycor != 0]`

34. Open het bestand *H2opg5.nlogo*. Je ziet vier buttons. Klik op alle buttons om hun effect te zien.
35. Beredeneer welke button (A, B, C of D) bij welke coderegels (i, ii, iii of iv) hoort.
36. Bekijk de coderegels van de procedure *setup* in figuur 2.10. Voorspel door middel van een schets wat je zal zien als deze code wordt uitgevoerd.
37. Ga naar het tabblad *code* en voeg de coderegels in figuur 2.10 toe aan de procedure *setup*.
38. Pas alleen het gedeelte `[pxcor >= 0 and pycor <= 0]` aan, zodanig dat na het uitvoeren van de *setup* figuur 2.11 verschijnt.

```
clear-all;
ask patches [
  set pcolor white
]
ask patches with
  [pxcor >= 0 and pycor <= 0] [
  set pcolor orange
]
```

FIGUUR 2.10



FIGUUR 2.11

Het is niet perse nodig om een getal in te vullen als eis.

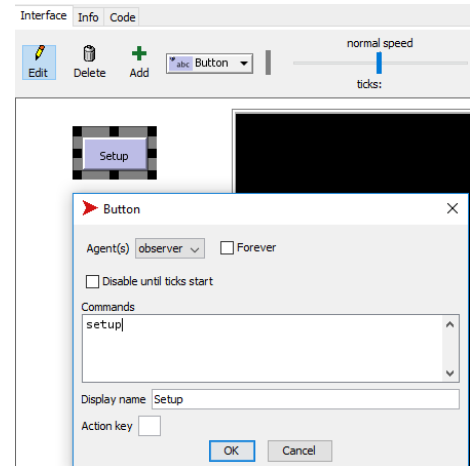
39. Welke patches voldoen aan de voorwaarden in onderstaande coderegels?
`ask patches with [pxcor = pycor or pxcor = -1 * pycor]`
40. Controleer jouw antwoord op de vorige vraag door de *setup* aan te passen.
41. Bedenk zelf een eis om patches te selecteren met *setup*, door gebruik te maken van logische operatoren in de voorwaarde. Vraag daarna je buurman of buurvrouw om een procedure te schrijven die hetzelfde resultaat oplevert (er zijn vaak meerdere mogelijkheden!).

Opdracht 6: Zelf een model maken



In deze opgave leer je om *from scratch* een NetLogo-model met patches te maken. Je maakt een modelwereld, voegt zelf een **button** toe en een **slider** toe.

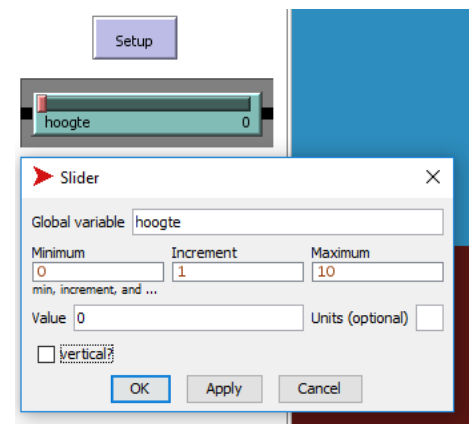
42. Open NetLogo en kies *File* → *New*. Er opent zich nu een leeg modelvenster met wereld.
43. Wat zijn de standaardinstellingen voor het aantal patches in deze wereld? Hoe groot worden de patches weergegeven?
44. Verander de modelwereld zodat deze uit uit 75×25 patches bestaat die met een grootte van 15 op het scherm worden getoond.
45. Klik op *Add* (+) en vervolgens in het witte vlak links naast de (zwarte) modelwereld. Er verschijnt nu een nieuwe button en een menu zoals in figuur 2.12.
Vul onder *Commands* de methodenaam *setup* in en bij *Display name* de naam *Setup*. (We kiezen er in deze module voor om dat steeds met hoofdletter *S* te schrijven, maar dat hoeft niet.) Klik als je klaar bent op *OK*.



FIGUUR 2.12

We hebben nu een button met *Setup*. De letters zijn rood, omdat de knop verwijst naar de methode *setup*, maar die bestaat nog niet! We gaan nu de bijbehorende procedure *setup* maken.

46. Ga naar het tabblad *Code* en maak een procedure *setup* die zorgt dat alle patches kleurcode 12 krijgen.
47. Pas de code aan zodat alle patches met een y-coördinaat groter of gelijk aan 0 kleurcode 95 krijgen (en alle andere patches nog steeds kleurcode 12).
48. Ga terug naar het tabblad *Interface* en klik naast *Add* (+) op het dropdown-menu met interface-items (Button) en selecteer de slider (Slider).
Neem alle waarden over uit figuur 2.13 en klik op *OK*.



FIGUUR 2.13

We hebben nu een schuifbalk (slider) gemaakt, waarmee we een variabele *hoogte* kunnen instellen. De bedoeling is dat we deze variabele gebruiken als **parameter** voor ons model. Een parameter kun je zien als (begin-) waarde voor het model. Je kunt hem wel veranderen voor je *setup* uitvoert om zo de beginsituatie aan te passen, maar als het model *runt*, verander je hem meestal niet.

Door de schuif te bewegen en daarna *setup* te runnen, veranderen we de invoer voor het model, zonder dat we daarvoor steeds de code moeten aanpassen. Hiervoor moeten we wel eenmalig de code herschrijven, zodat daarin gebruik gemaakt wordt van de parameter.

49. Verander in de code van *setup* de regel met het fragment " $> = 0$ " naar " $> = \text{hoogte}$ ".
50. Verander meerdere keren de positie van de slider. Klik klik tussendoor steeds op de *Setup*-button, want anders wordt er niets met de ingestelde waarde gedaan.

De instellingen zijn op dit moment zodanig, dat alleen het bovenste gedeelte van de modelwereld blauw kan worden. Het moet mogelijk worden om de situatie uit figuur 2.14 te krijgen.

51. Pas de minimale en maximale waarde van de slider aan, zodat exact kan worden ingesteld dat de wereld helemaal blauw (kleurcode 95) is of helemaal donkerrood (kleurcode 12) wordt.
52. Welke waarden heb je nu ingevuld bij het *Minimum* en het *Maximum*? Controleer of dit echt precies de grenswaarden zijn.
53. Kies *File* → *Save as...* om het bestand te bewaren.



FIGUUR 2.14

Opdracht 7: if, else & random

In Hoofdstuk 1 heb je kunnen zien dat het begrip **random** (willekeur) erg belangrijk is als je gaat modelleren. Bij opdracht 2.1 hebben we gezien dat de bomen in het bosbrand-model niet altijd precies op dezelfde plek staan en dat hierdoor de uitkomst van het model per run kan verschillen.

Hoe voeg je willekeur (random) toe aan een model? Om dat uit te leggen kijken we eerst hoe je **keuze** of een **voorwaarde** kunt inbouwen met **if** en **ifelse**.

54. Open het bestand *H2opg7.nlogo*. Je ziet drie buttons. Klik op alle buttons om hun effect te zien.

De button met *if* erop, verwijst naar de procedure in figuur 2.15 en de button met *ifelse* naar de procedure in figuur 2.16. Het werken met *if* en *else* ken je vast van eerdere programmeerlessen. Let vooral goed op de manier waarop de code in NetLogo moet worden geschreven!

55. De button met *combinatie* runt de code van allebei de procedures. Bedenk door naar het resultaat te kijken of hier eerst de *ifelse* wordt uitgevoerd en daarna de *if* of andersom.

NetLogo heeft een functie **random**. De regel

```
set pcolor random 140
```

levert een kleurcode op van 0 tot 140. De 140 zelf doet dus niet mee!

56. Verander de regel `set pcolor 107` in bovenstaande regel met `random`. Klik daarna meerdere keren op de *if*-button.

We gebruiken de `random`-functie nu voor een willekeurige kleur. Maar hoe kunnen we willekeurig patches vragen om iets te doen?

57. Kopieer de code van *if_else_procedure*, plak de code en verander de naam van de procedure in *random_procedure*.
58. Verander in deze nieuwe procedure de regel `ifelse pycor > 0` in `ifelse random 100 < 50`.
59. Voorspel wat het resultaat zal zijn van deze nieuwe procedure?
60. Voeg een button met de naam *random* toe die de nieuwe procedure aanroept. Klik vervolgens op de button om jouw voorspelling van de vorige vraag te controleren.
61. Stel dat we de waarde `50` veranderen in `10`: zien we dan meer of minder roze patches?
62. Maak een slider *percentage* die kan variëren tussen de 0 en de 100. Pas de code aan, zodat de procedure niet meer de waarde `50` gebruikt, maar reageert op het slider-percentage.

Opdracht 8: Aquarium I | patches als algen

Je gaat, verspreid over meerdere opdrachten, een model maken volgens de volgende casus *Aquarium*:

Een groot aquarium in een dierentuin bevat vissen die willekeurig rondzwemmen. Zwemmen kost energie. De vissen kunnen hun energie aanvullen door in het water aanwezige algen te eten. Vissen kunnen zich voortplanten als ze voldoende energie hebben en sterven als ze geen energie meer hebben. Binnen het aquarium verandert de hoeveelheid vissen voortdurend.

Voor de algen geldt iets vergelijkbaars: ze worden gegeten door de vissen, maar er is ook algengroei.

63. Open het bestand *H2opg8_AQ1.nlogo*.
64. Maak een procedure *setup* die ervoor zorgt dat elke patch met een kans van 10% groen (*green*) wordt of anders blauw (*blue*). De groene patches stellen de algen voor.
65. Voeg een button *Setup* toe om te controleren of je het goed hebt gedaan.
66. Denk goed na: is het nu altijd zo dat precies 10% van de patches groen is na *setup*?

```
to if_procedure
  clear-all
  ask patches [
    if pxcor > 0
    [
      set pcolor 107
    ]
  ]
end
```

FIGUUR 2.15

```
to if_else_procedure
  clear-all
  ask patches [
    ifelse pycor > 0
    [
      set pcolor 18
    ]
    [
      set pcolor 27
    ]
  ]
end
```

FIGUUR 2.16

2.4 Bewegende agents

In hoofdstuk 1 heb je allerlei modellen kunnen zien waarin agents door de modelwereld bewegen. Patches hebben een vaste plaats: ze kunnen dus niet bewegen. Voor bewegende agents zijn er in NetLogo **turtles**. Ze danken deze naam (*turtle* is Engels voor schildpad) aan het feit dat de programmeertaal NetLogo vroeger werd gebruikt om een schildpad over het scherm te laten bewegen (later ook een robotschildpad).

Net als bij een patch kun je aan turtles **attributen** toevoegen, naast de standaard-eigenschappen die door NetLogo aan elke turtle worden gegeven. Maar waar patches al bestaan omdat we een modelwereld hebben, moeten de turtles nog worden gemaakt.

In figuur 2.17 zie je een setup-procedure waarin twee turtles worden gemaakt met de opdracht `create-turtles`. De turtles krijgen een plek in de modelwereld met de functie `setxy` waaraan je eerst een x- en daarna een y-coördinaat meegeeft. In dit geval is gekozen voor een willekeurige plek met `random-xcor` en `random-ycor`, maar je kan hier ook getallen invoeren.

```
to setup
  clear-all
  reset-ticks

  create-turtles 2 [
    setxy random-xcor random-ycor
    set color random 140
  ]
end
```

FIGUUR 2.17

Als we nu de setup uitvoeren, verschijnen er twee turtles, maar ze bewegen nog niet. De setup is immers alleen een **initialisatie**: het instellen van een beginsituatie. Om de turtles te laten bewegen, maken we een procedure **go**. Dit is de standaardnaam voor de hoofdprocedure (het hoofdprogramma) van een model. Deze bevat alle stappen die horen bij één **iteratie** (zie § 1.3) van het model.

Nadat eenmalig de *setup* is uitgevoerd, wordt de procedure *go* normaal gesproken steeds opnieuw uitgevoerd. Elke keer dat dit gebeurt, verstrijkt er een klein stap in de tijd die in NetLogo standaard wordt aangeduid met tick of **ticks**. Het aantal ticks geeft daarmee het aantal iteraties van het model. Als je het model opnieuw wil uitvoeren, begin je met een nieuwe *setup*. De tijd wordt weer op 0 gezet met `reset-ticks`.

figuur 2.18 toont een eenvoudig voorbeeld van een *go*-procedure die de turtles laat bewegen. Net als bij patches vragen we de turtles om iets te doen met `ask`. In dit geval geven we twee opdrachten:

- `rt random 360`
Draai naar rechts (*right turn*) over een willekeurige hoek tussen de 0 en 360 graden. (`rt` (*right turn*) kan natuurlijk ook!)
- `fd 1`
Beweeg vooruit (*forward*) met een stapje van 1.

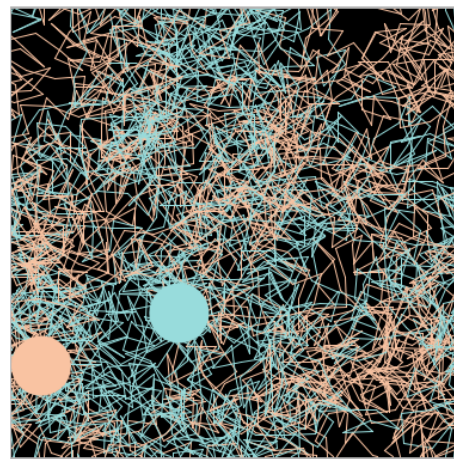
```
to go
  ask turtles [
    rt random 360
    fd 1
  ]
  tick
end
```

FIGUUR 2.18

Opdracht 9: Random walk

Een *random walk* of toevalsbeweging is een beweging volgens een pad met willekeurige stappen. In de natuurkunde en scheikunde zien we dit terug bij de beweging van kleine deeltjes (Brownse beweging, diffusie), maar het begrip komt ook terug binnen bijvoorbeeld economie en (andere onderdelen van) informatica.

67. Open het bestand *H2opg9.nlogo*. Dit model bevat de code uit de theorie hierboven. Klik meerdere keren op *Setup*. Hoe kun je zien dat de modelwereld *wraparound* is?
68. Klik op *Go*. De turtles gaan nu bewegen. Bovenaan zie je het aantal ticks. Verschuif de slider tot \pm één tick per seconde.
69. Voeg één voor één de volgende regels toe aan de `create-turtles` –opdracht en bekijk steeds tussendoor het resultaat:
 - `set shape "circle"`
 - `set size 2`
 - `pen-down`



FIGUUR 2.19

Opdracht 10: Meer attributen | stappen tellen



In deze opgaven kijken we naar een aantal standardeigenschappen en leer je hoe je zelf nieuwe attributen aan turtles toe kunt voegen.

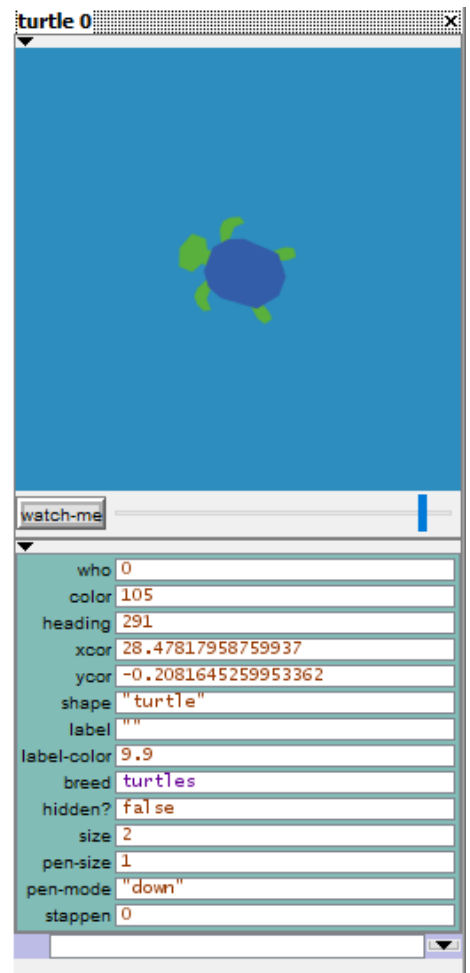
70. Open het bestand *H2opg10.nlogo*. Klik vervolgens op *Setup* en daarna op *Go*. Je ziet twee schildpadden die willekeurig door de ruimte lopen.
71. Klik nogmaals op *Go* om het model te onderbreken.
72. Klik met de rechtermuisknop op een schildpad en kies *Turtle 0* (of 1) → *inspect*. Er opent zich een venster vergelijkbaar met figuur 2.20. Welke attributen heeft een turtle standaard?
73. In hetzelfde menu had je in plaats van *inspect* ook kunnen kiezen voor *watch* of *follow*. Probeer ze allebei uit en beschrijf de verschillen tussen de drie functies.

Het attribuut *stappen* in figuur 2.20 is geen standaard-eigenschap.

74. Ga naar het tabblad *Code*:
 - Met welke coderegel is dit attribuut toegevoegd?
 - Met welke regel wordt het aantal stappen verhoogd?
75. Haal de `;` weg voor de regel `set label stappen` om deze te activeren. Bekijk het resultaat.

De agents krijgen de vorm van een schildpad door `set shape "turtle"`. Maar hoe weet je welke vormen je zou kunnen gebruiken?

76. Kies in het menu voor *Tools* → *Turtle Shapes Editor*. Kies een andere *shape* en verander de code zodat deze wordt gebruikt.
77. Zet een `;` voor de regels `set shape` en `pen-down`. Wat is de standaardvorm van een turtle in een NetLogo-model?
78. Gebruik een voorwaarde (*if*) om de turtles een rode kleur te geven als ze 10 of meer stappen hebben gezet.



FIGUUR 2.20

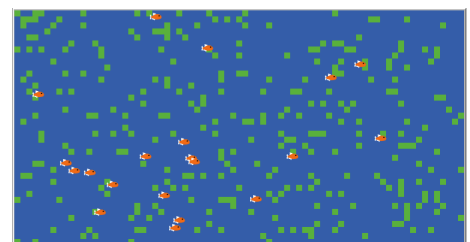
Opdracht 11: Aquarium II | turtles als vissen

We gaan ons model van het aquarium uitbreiden. Als uitgangspunt nemen we het eindresultaat van opdracht 8.

79. Open het bestand *H2opg11_AQ2.nlogo*. Klik vervolgens op *Setup*. De patches kleuren met een kans van 10% groen. Dat zijn de algen in het aquarium.

We gaan nu turtles maken die als vissen (agents) zullen dienen.

80. Breid de *setup* uit zodat er 20 turtles worden gemaakt die op een willekeurige plek in de modelwereld worden geplaatst. Geef ze als vorm (*shape*) "fish", grootte 2 en kleur oranje.
81. Geef de turtles een nieuw attribuut *energie*.
82. Zorg dat de alle vissen (*turtles*) beginnen met een *energie* van 10.
83. Voeg een *go*-procedure toe om de vissen willekeurig te laten zwemmen. (Hint: zie figuur 2.18)
84. Klik op de *Go*-button om te verifiëren of alles werkt zoals je het verwacht. Als het goed is lijkt je model nu op figuur 2.21.



FIGUUR 2.21

Opdracht 12: Resterende stappen | stop en sterf

Bij de vorige opgave kregen de vissen een attribuut *stappen* met in eerste instantie de waarde 0. Bij elke iteratie (*tick*) werd het aantal stappen met één verhoogd. We doen nu het omgekeerde in een model zonder vissen maar met turtles:

Turtles krijgen een random aantal stappen toegewezen dat bij elke iteratie één kleiner wordt. Wanneer een schildpad geen stappen meer over heeft sterft hij. Zodra een schildpad vijf stappen of minder te gaan heeft krijgt hij een rode kleur.

85. Open het bestand *H2opg12.nlogo*. Klik vervolgens op *Setup*. Stel vast dat de turtles niet allemaal hetzelfde aantal *stappen* hebben.
86. Open het tabblad *Code* en bestudeer de coderegels. Merk op dat er meerdere nieuwe procedures (*random_walk*, *controleer_stappen*) zijn gemaakt om de *go*-procedure overzichtelijk te houden.
87. Wat is het aantal stappen waarmee een turtle kan starten na het uitvoeren van de *Setup*? Hoe ben je daar achter gekomen?
88. Ga terug naar de *Interface* en klik op *Setup* en daarna een paar keer op *1x Go*. Wat gebeurt er?
89. Klik met de rechtermuisknop op de button *1x Go* en kies *Edit....* Herhaal dit voor de button *Go*. Wat is het (enige) verschil qua instellingen tussen beide buttons?

Als we het model runnen door op *Go* te klikken, blijft het steeds doorgaan. Vaak wil je dat het model zelf onder bepaalde voorwaarden stopt. Zo'n voorwaarde heet een **stopconditie**.

```
if ticks = 20 [  
  stop  
]
```

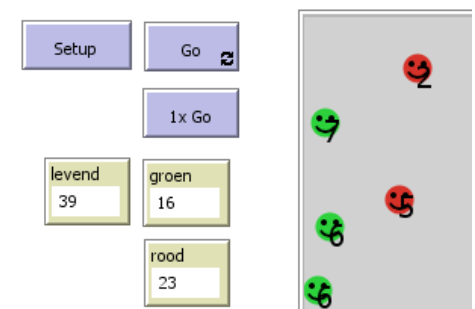
FIGUUR 2.22

90. Voeg de stopconditie van figuur 2.22 toe aan *go*, boven de coderegel *tick* en bekijk het resultaat.
91. Voer de volgende stappen uit
 - Klik op *Setup*
 - Selecteer een turtle met meer dan 20 stappen en kies (rechtermuisknop) *inspect*.
 - Run het model door op *Go* te klikken. Hoeveel stappen heeft de turtle gelopen?
92. Verplaats de stopconditie (alle coderegels uit figuur 2.22) naar het begin van de procedure *Go*. Herhaal vervolgens de drie stappen van de vorige vraag. Wat is je conclusie?

Bij de vorige opdracht kregen de turtles een rood hoofd na 10 stappen. Dit zorgde ervoor dat alle turtles bij dezelfde iteratie van kleur veranderden. De procedure *controleer_stappen* is nu nog leeg, maar moet zorgen dat aan de volgende eis wordt voldaan: *Bij vijf stappen of minder krijgt de turtle een rode kleur*.

93. Gebruik een voorwaarde om bovenstaande eis in het model in te bouwen. Natuurlijk controleer je de juist werking. Hoe heb je deze verificatie aangepakt?

Het model bevat twee monitoren *levend* en *groen*. Met een **monitor** kun je een actuele waarde van een model of een berekening laten zien Van een variabele of een berekingen in het model. In figuur 2.23 zie je de monitoren, inclusief een monitor *rood*.



FIGUUR 2.23

94. Klik met de rechtermuisknop op de monitor *levend* en kies *edit*. Op welke manier wordt hier het aantal turtles getoond?
95. Herhaal bovenstaande stappen voor de monitor *groen*.
96. Klik naast *Add (+)* op het dropdown-menu met interface-items (*Button*) en selecteer de monitor (*Monitor*). Voeg een monitor *rood* toe die het aantal turtles met een rood hoofd telt. Controleer de werking.

Turtles kunnen op dit moment nog steeds een negatief aantal stappen hebben. De laatste eis luidt:

als een turtle 0 resterende stappen heeft, sterft hij.

97. Voeg de code uit figuur 2.24 toe aan de procedure *controleer_stappen*. Bekijk het resultaat.
98. Verhoog het aantal turtles naar 100 en voer het model uit. Merk op dat de monitoren in dit geval percentages aangeven.

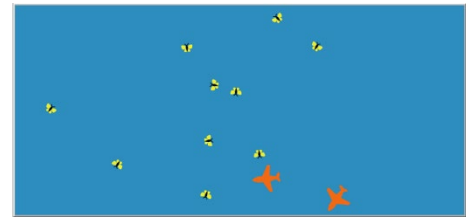
```
if stappen = 0 [  
  die  
]
```

FIGUUR 2.24

Opdracht 13: Breeds: soorten turtles

Het voelt misschien een beetje vreemd om mensen of vissen te modelleren als *turtles*. En wat nu als je een model wilt maken met zowel mensen als vissen? Hoe maak je dan onderscheid?

In figuur 2.25 zie je twee **soorten** turtles of **breeds**. Met een *breed* (Engels: ras of soort) kun je een aparte soort turtles maken met specifieke eigenschappen. Door ze bij hun eigen naam te noemen kun je elke soort apart opdrachten geven, in plaats van alle turtles tegelijk.



FIGUUR 2.25

Een *breed* maak je helemaal bovenaan je programma, met de regel:
`breed [mensen mens]`

Na het commando `breed` geef je dus zowel de meervouds- als de enkelvoudsvorm van jouw nieuwe soort. In de rest van het programma kun je bij alle coderegels waar je eerst *turtle* of *turtles* gebruikte nu kiezen voor (in dit voorbeeld) *mens* en *mensen*.

99. Open het bestand *H2opg13.nlogo*. Klik vervolgens op *Setup* en *Go* om de uitvoer te zien.
100. Bestudeer de code van de procedure *maak_vliegtuigen*. Een vliegtuig krijgt het voor ons nieuwe attribuut `heading` mee. Zoek uit wat je met deze eigenschap instelt.
101. De vliegtuigen maken geen *random walk*. Beschrijf op basis van de code in de *Go*-procedure in algemene termen op welke manier ze bewegen. Wat gebeurt er in één iteratie?
102. Maak een nieuwe soort turtles *vlinders*.
103. Maak een procedure *maak_vlinders* die 10 gele vlinders met grootte 2 op een willekeurige plek in de modelwereld plaatst (zie figuur 2.25).
104. Breid de *Go*-procedure uit, zodat de vlinders vliegen volgens het principe van de *random walk*.

Opdracht 14: Aquarium III | vissen als breed

Met de kennis van de vorige opgave, kunnen we ons aquarium-model aanpassen met een *breed* vissen. We willen het volgende bereiken:

- Er is een *breed* vissen. In de code wordt alleen verwezen naar *vissen*; het woord *turtle* komt niet meer voor.
- Er zijn aparte procedures voor het maken van de patches en de vissen, zodat de setup gelijk is aan de code in figuur 2.26.
- Vissen hebben nog steeds het attribuut *energie*. Omdat vissen nu een soort turtles is, gebruiken we hiervoor de code:

```
vissen-own [  
  energie  
]
```

105. Open het bestand *H2opg14_AQ3.nlogo*. Dit is het eindresultaat van onze vorige aquarium-opgave.
106. Pas het model aan zodat aan bovenstaande eisen wordt voldaan.

Zwemmen kost energie. Bij elke iteratie van het model moet de energie van de vissen met 1 afnemen.

107. Maak een aparte procedure *zwem* zodat de *go*-procedure alleen nog bestaat uit de code in figuur 2.27.
108. Pas de procedure *maak_vissen* aan zodat de vissen een willekeurige begin-energie krijgen van minimaal 10 en maximaal 100. Geef de vissen een label mee, zodat je kunt controleren (verificatie) of je het goed hebt gedaan.
109. Pas de procedure *zwem* aan, zodat de energie van de vissen bij elke tick met 1 afneemt.
110. Pas het model aan, zodat de vissen dood gaan wanneer hun energie 0 is. Voeg een monitor *aantal levende vissen* toe.
111. Breid de *go* uit, zodat het model stopt als er 0 vissen zijn.

```
to setup  
  clear-all  
  reset-ticks  
  
  maak_algen  
  maak_vissen  
  
end
```

FIGUUR 2.26

```
to go  
  ask vissen [  
    zwem  
  ]  
  tick  
end
```

FIGUUR 2.27

2.5 Agents die reageren op hun omgeving

In de modellen in hoofdstuk 1 hebben we kunnen zien dat agents doorgaans reageren op hun omgeving. In het model dat de *evacuatie van mensen uit een voetbalstadion* simuleerde, was het gedrag van de mensen afhankelijk van gangen en wanden (vaste objecten: patches) en andere mensen in de buurt (andere, bewegende agents: turtles). De regels die we voor agents programmeren, moeten er dus voor zorgen dat hun gedrag afhangt van de patches en turtles in het model. We kunnen dit opdelen in drie categorieën:

- patches die reageren op andere patches of ze beïnvloeden
- turtles die reageren op andere turtles of ze beïnvloeden
- turtles en patches die op elkaar reageren of elkaar beïnvloeden

In figuur 2.28 zie je linksboven een aantal patches in een beginsituatie. Een aantal patches is zwart gemaakt bij de setup. De patches hebben geteld hoeveel zwarte burenen ze hebben (zie het oranje label) en dit bewaard in een attribuut burenen met de code:

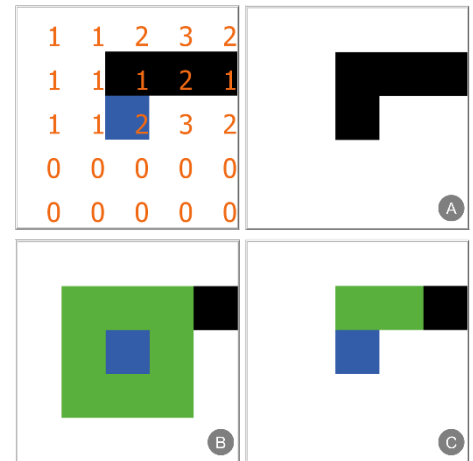
```
set burenen count neighbors with [pcolor = black]
```

Met `count neighbors` kunnen we het aantal burenen tellen. Door de toevoeging `with` kunnen eisen stellen aan de patches die we willen tellen.

Voor de blauwe patch in figuur 2.28 geldt dat hij twee zwarte burenen heeft. De blauwe patch heeft hier in figuur ^A op gereageerd, door zwart te kleuren met de `go`-procedure in figuur 2.29.

In figuur 2.28 ^B heeft de blauwe patch juist aan zijn burenen gevraagd om te veranderen met `ask neighbors`. Alle burenen zijn nu groen gekleurd. De bijbehorende code zie je in figuur 2.30. In figuur ^C hebben alleen de burenen die zelf zwart gekleurd waren gereageerd. Hiervoor is wederom de toevoeging `with` gebruikt:

```
ask neighbors with [pcolor = black] [
  set pcolor green
]
```



FIGUUR 2.28

```
to go
  ask patch 2 2 [
    if burenen > 1 [
      set pcolor black
    ]
  ]
end
```

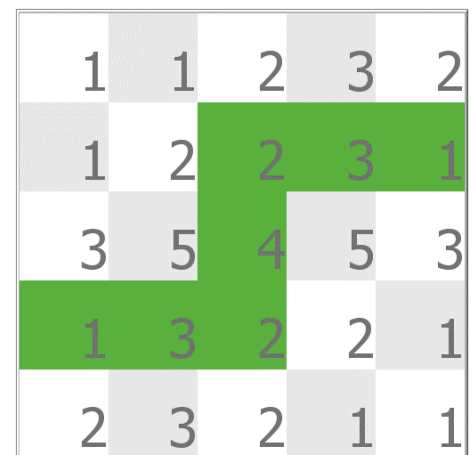
FIGUUR 2.29

```
to go
  ask patch 2 2 [
    ask neighbors [
      set pcolor green
    ]
  ]
end
```

FIGUUR 2.30

Opdracht 15: de juiste vraag-volgorde

112. Open het bestand *H2opg15.nlogo*. Klik vervolgens op *Setup*. Je ziet nu het beeld uit figuur 2.31. Net als in de theorie hierboven geeft het *plabel* het aantal gekleurde (groene) burenen.
113. Leg uit hoe je kunt zien wat de instelling voor world wrap is.
114. Klik op *1x Go* totdat alle patches groen zijn en concentreer je op hoe het patroon verandert. Klik vervolgens op *Setup* en herhaal het klikken op *Go*. Zie je verschillen?
115. Ga naar het tabblad *Code*. In de `go`-procedure wordt aan de patches gevraagd om achtereenvolgens `verander_kleur` en `tel_burenen` uit te voeren. Pas de code aan zodat eerst aan alle patches wordt gevraagd om van kleur te veranderen en daarna (in een aparte `ask patch`) om burenen te tellen.
116. Stel vast dat het patroon nu wèl steeds volgens dezelfde stappen verandert en net niet. Wat is de verklaring?



FIGUUR 2.31

Opdracht 16: Patches en hun omgeving

117. Open het bestand *H2opg16.nlogo*. Dit is het eindresultaat van de vorige opdracht.

118. Breid de procedure `verander_kleur` uit, zodat een patch blauw (*blue*) kleurt, als het aantal burenen groter dan 4 is. (De patch moet nog steeds groen worden bij meer dan 2 burenen!) Bekijk het resultaat.

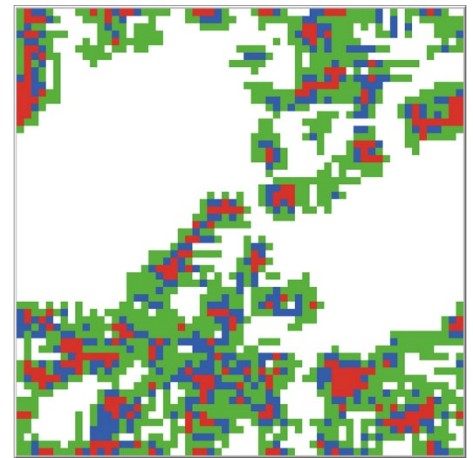
119. In de *go*-procedure is de procedure `kleur_burenen` uitgeschakeld (met `;`) omdat deze nog niet bestaat. Maak een procedure `kleur_burenen` die ervoor zorgt dat patches aan hun burenen vragen om de kleur rood (*red*) aan te nemen als deze burenen zelf meer dan 5 burenen hebben.

Controleer het resultaat door de `;` in de *go* weg te halen.

120. Nu we de stappen op kleine schaal hebben gecontroleerd, kunnen we onze wereld vergroten.

Voer de volgende stappen uit:

- Gebruik een puntkomma om te voorkomen dat de patches een *plabel* weergeven.
- Vergroot de wereld naar 60×60 patches met een grootte van 10.
- Voeg een *Go*-button toe die steeds opnieuw de *Go*-procedure uitvoert en bekijk het eindresultaat.



FIGUUR 2.32

Opdracht 17: Werkblad iteraties H1



In hoofdstuk 1 heb je misschien het werkblad *iteraties* gemaakt waarbij cellen (die we nu patches zouden noemen) zwart of wit kleuren op basis van de regel:

Als er een even aantal ingekleurde burenen is, is de kleur van het veld bij de volgende stap zwart. Bij een oneven aantal is de kleur van het veld bij de volgende stap wit.

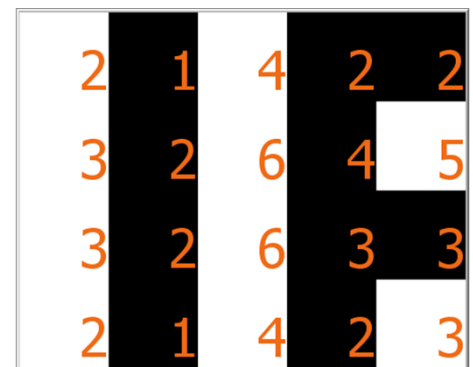
In figuur 2.33 zie je boven de beginsituatie en onder het resultaat nadat de regel hierboven één keer is uitgevoerd.

121. Het oranje label geeft het aantal zwarte burenen van een patch aan. Is de modelwereld in figuur 2.33 *wraparound* of niet?
122. Stel voordat je verder gaat vast dat je begrijpt dat het uitvoeren van de regel leidt tot de onderste figuur van figuur 2.33.
123. Open het bestand *H2opg17.nlogo*. Klik vervolgens op *Setup*.
124. Vul de procedure `tel_burenen` aan, zodat deze de eigenschap `burenen` van alle patches vult met het aantal zwarte buurpatches.
125. Voer de *setup* uit en controleer of je hetzelfde resultaat krijgt als in het linker plaatje van figuur 2.33.

Om de modelregel te simuleren in NetLogo moeten we vaststellen of het aantal burenen even of oneven is. Dit kan met:

`if burenen mod 2 = 0 [; vul aan met modelregels voor als dit waar is]`

126. Vul de procedure `verander_kleur` aan, zodat alle patches van kleur veranderen volgens de gegeven regel. Gebruik hierbij de operator `mod`. (Is `mod` nog onbekend voor je? Zoek het op!)
127. Voer het model uit en controleer of het model na één iteratie het resultaat uit figuur 2.33 geeft.
128. Het model komt na een aantal iteraties weer in de beginsituatie. Na hoeveel iteraties is dat?
129. Pas de modelwereld aan zodat *world wrap* aan staat. Komt het model nog steeds terug naar de beginsituatie?



FIGUUR 2.33

Opdracht 18: Aquarium IV | groeiende algen

In de opdrachten in deze paragraaf heb je kunnen oefenen met patches die reageren op hun omgeving, maar de voorgaande modellen hadden geen betekenis of context. In ons aquarium-model stellen de patches de algen voor. Algen groeien.

We willen het volgende bereiken:

- Algen hebben elke tick een kans om te groeien.
- Groeien betekent dat één van de omliggende patches van blauw in groen verandert.
- Een alg kan alleen groeien, als zijn omgeving niet volledig gevuld is met algen, ofwel: er moet wel een blauwe patch naast een groene patch zitten.

```
to groei_algen
  ask patches [
    if count (neighbors with [pcolor
= blue]) > 0 [
      ask one-of neighbors with
[pcolor = blue] [
        set pcolor yellow
      ]
    ]
  ]
end
```

FIGUUR 2.34

130. Open het bestand *H2opg18_AQ4.nlogo* en klik op *Setup*.

131. Het model bevat een monitor *% algen*. Bekijk de eigenschappen van deze monitor. Hoe wordt het percentage algen hier berekend? Stel vast dat je de bijbehorende code begrijpt.

In de code is een procedure *groei_algen* gemaakt (zie ook figuur 2.34). Deze bevat de code-regel:

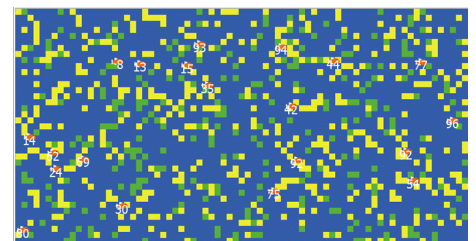
ask one-of neighbors with [pcolor = blue]

Met *one-of* vraag je aan één van de agents die aan een eis voldoet (in dit geval patchkleur blauw) om iets te doen.

132. Wat is de betekenis van de regel *if count (neighbors with [pcolor = blue]) > 0* ?

133. Klik op *1x Go* zodat de procedure *go* één keer wordt uitgevoerd. Wat is het resultaat?

Er zijn heel veel gele patches bijgekomen. (Er is voor de kleur geel gekozen, zodat je in deze fase het verschil ziet tussen nieuwe en bestaande algen.) Dit komt, omdat nog niet aan de eerste eis is voldaan.



FIGUUR 2.35

134. Pas de procedure *groei_algen* aan, zodat er een kans van 2% is dat een blauwe patch geel wordt, als hem iets wordt gevraagd. Bekijk het resultaat door een aantal keren op *1x Go* te klikken.

135. Verklaar dat de monitor *% algen* niet mee verandert.

136. Maak een slider *groeikans* (minimaal 0, maximaal 50) die de kans geeft dat een alg gaat groeien.

Pas de code aan, zodat de procedure *groei_algen* gebruik maakt van de parameter *groeikans*.

137. Werkt alles naar behoren? Verander dan *yellow* in *green*. Werkt de monitor nu wel?

Opdracht 19: Game of Life



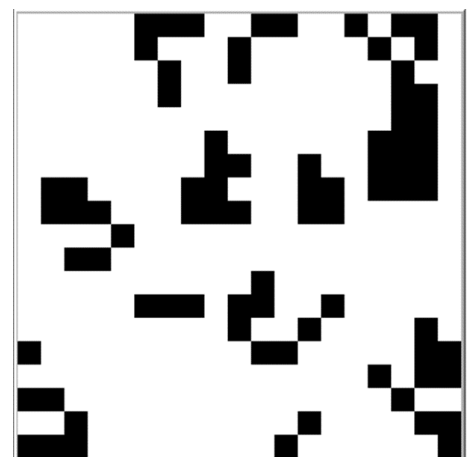
In hoofdstuk 1 heb je misschien al kennis gemaakt met de *Game of Life*. Er bestaat ook een 3D-versies, zoals: <http://cubes.io>

138. Open het bestand *H2opg19.nlogo* en klik op *Setup*. Je ziet een wereld met levende (zwarte) en dode (witte) cellen.

139. Pas de parameter *percentage_levend* een aantal keren aan en klik tussendoor steeds op *Setup*. Verklaar waarom de monitor *% levend* (het percentage levende cellen) niet altijd dezelfde waarde geeft als de met de slider ingestelde parameter.

140. Zoek met internet (en / of hoofdstuk 1) op wat *Game of Life* is en volgens welke regels de patches in de modelwereld (blijven) leven of sterven.

141. Patches krijgen in *go* de opdracht *voer_regels_uit*. Deze procedure is op dit moment nog leeg. Programmeer de regels van *Game of Life* binnen deze procedure.



FIGUUR 2.36

2.6 Turtles die op elkaar reageren

In de vorige paragraaf hebben we gezien hoe we patches op hun omgeving kunnen laten reageren. We gaan kijken hoe turtles elkaar kunnen beïnvloeden aan de hand van de volgende casus:

In een groep mensen is één persoon die een nieuwe mop kent. Hij vertelt de mop één keer door aan iemand in zijn buurt. Die vertelt op zijn beurt de mop ook één keer door, totdat de mop wordt verteld aan iemand die alleen maar burens heeft die de mop al kennen (en er niemand is om de mop aan door te vertellen). Hoe verspreidt de mop zich over de groep? Hoeveel mensen krijgen de mop te horen?

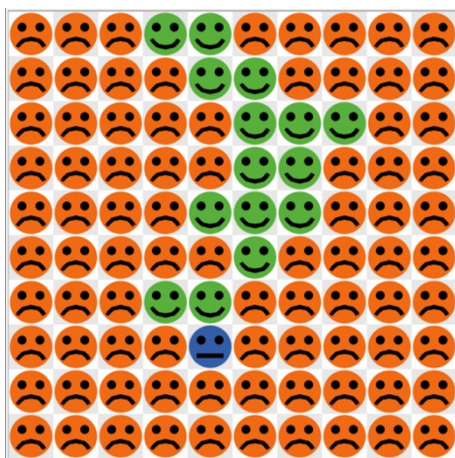
Voor ons model gebruiken we de `breed` mensen die de attributen `verteller?` en `kentMop?` meekrijgen met `mensen-own`. Attributen met een `?` zijn van het type **boolean** dat je misschien kent van de programmeerlessen. Ze zijn waar (`true`) of onwaar (`false`).

In figuur 2.37 zie je de setup-procedure van dit model. Deze bevat code die we nog niet eerder hebben gezien:

- `sprout-mensen 1`
Hiermee vragen we de patches om op de patch een aantal (1) turtles van het type `mens` te maken (gevolgd door attributen).
- `ask one-of` mensen
Als de patches mensen hebben gemaakt, vragen we één van deze mensen om `verteller?` te worden.

Het resultaat van de code is een modelwereld met oranje mensen die de mop nog niet kennen en één blauwe verteller die de mop wel kent. Deze verteller moet de mop nu gaan doorvertellen aan één van de mensen in zijn omgeving die de mop nog niet kent. We willen dat de verteller zelf groen wordt als hij de mop heeft doorverteld.

De procedure `vertel_mop` in figuur 2.38 voldoet aan deze eisen. Na een aantal iteraties geeft dit een resultaat zoals in figuur 2.39.



FIGUUR 2.39

```
breed [mensen mens]

mensen-own [
  verteller?
  kentMop?
]

to setup
  clear-all
  reset-ticks
  ask patches [
    sprout-mensen 1 [
      set verteller? false
      set kentMop? false
      set color orange
      set shape "face sad"
    ]
  ]

  ask one-of mensen [
    set kentMop? true
    set verteller? true
    set color blue
    set shape "face neutral"
  ]
end
```

FIGUUR 2.37

```
to vertel_mop
  if (count (mensen-on
  neighbors) with [not kentMop?]
  > 0) [
    ask one-of (mensen-on
  neighbors) with [not kentMop?]
  [
    set kentMop? true
    set verteller? true
    set color blue
    set shape "face neutral"
  ]
  set color green
]
set verteller? false
set shape "face happy"
end
```

FIGUUR 2.38

Merk op dat `vertel_mop` eerst controleert of de verteller nog wel burens heeft die de mop niet kennen. Dit voorkomt dat NetLogo de opdracht `ask one-of` krijgt, terwijl er niemand meer in zijn directe omgeving is die voldoet aan de eis dat hij de mop nog niet kent.

Deze paragraaf geeft slechts één voorbeeld van hoe turtles elkaar kunnen beïnvloeden. In de volgende opdrachten volgen meer methodes, zoals bewegende turtles die elkaar ontmoeten.

Opdracht 20: Een mop doorvertellen I

142. Open het bestand *H2opg20.nlogo*. Dit is het volledige model uit de theorie van deze paragraaf.
143. Run het model enkele keren en bekijk steeds de eindwaarde van de monitor *niet gelachen*. Zit er veel variatie in het aantal mensen dat de mop niet te horen krijgt? En in het aantal iteraties?

In de *setup* wordt één mens gevraagd om verteller te worden met *ask one-of* mensen. Als je aan meer dan één turtle (maar niet aan alle) iets wilt vragen, kun je hiervoor *ask n-of 2* mensen gebruiken met het door jou gewenste aantal (hier 2).

144. Voorspel wat de invloed zal zijn op het aantal iteraties van het model en het aantal mensen dat de mop niet te horen krijgt, als we het model niet met één maar met drie vertellers zouden beginnen.
145. Pas het model aan, zodat er in het begin drie vertellers zijn en controleer je voorspelling.

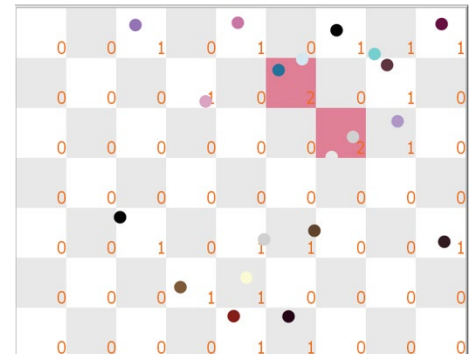
Onze modelwereld is erg vol. Er zijn 100 patches en overall staat een mens: de *bevolkingsdichtheid* = 100%. We willen het mogelijk maken dat deze bevolkingsdichtheid instelbaar is. Het gevolg hiervan is, dat de mensen niet meer standaard acht burens hebben.

146. Hoe denk jij dat het percentage mensen dat de mop niet te horen krijgt (*% niet gelachen*) afhangt van de *bevolkingsdichtheid*?
Teken een grafiek op papier met op de horizontale as de parameter *bevolkingsdichtheid* en op de verticale as het *% niet gelachen* en schets hierin een curve die jouw voorspelling weergeeft van het verband tussen de parameter en de uitkomst van het model.
147. Voeg een slider *bevolkingsdichtheid* toe en pas de *setup* aan, zodat je de kans kunt instellen dat een patch de opdracht *sprout_mensen* uitvoert. Controleer de juiste werking.
148. Pas de monitor *niet gelachen* aan naar *% niet gelachen*. Welke berekening is hiervoor nodig?
149. Controleer of jouw voorspelling bij vraag 146 overeenkomt met de uitkomsten van het model.

Opdracht 21: Random ontmoeting

In de vorige opdracht hebben we turtles laten communiceren die elkaars burens waren. We kunnen er ook voor kiezen dat turtles alleen met elkaar communiceren als ze op dezelfde plek zijn. Maar wanneer is dat het geval?

150. Open het bestand *H2opg21.nlogo*. Dit model is een uitbreiding op het *random walk*-model van opdracht 9.
151. Klik op *Setup*. De patches hebben een oranje label dat aangeeft hoeveel turtles zich op de patch bevinden. Zoek uit met welke coderegel dit aantal wordt vastgesteld.
152. Klik een aantal keren op *1x Go*. Er worden steeds andere patches roze (zoals in figuur 2.40). Wat is de betekenis van een roze patch?
153. Patches hebben een eigenschap die aangeeft hoe vaak er een ontmoeting heeft plaatsgevonden op die patch. Pas het label van de patches aan, zodat dit aantal wordt getoond. Controleer vervolgens de werking.
KIJK GOED: je moet misschien één programmeerstap meer doen dan je denkt! (Hint: staat het aantal ontmoetingen van een roze patch na de setup bij jou op 1 of op 0?)
154. Breid het model uit zodat alle turtles een attribuut ontmoeting krijgen die bijhoudt hoe vaak een turtle een andere turtle heeft ontmoet. Vergeet niet de beginwaarde op 0 te zetten.
155. Gebruik de code in figuur 2.41 om de patches aan de turtles te laten vragen om bij elke ontmoeting de bijbehorende variabele op te hogen. Waar moet deze code worden geplaatst?
156. Geef de turtles een blauw label mee dat de waarde van ontmoeting toont. Controleer hiermee de juiste werking van jouw model.
157. Bouw een stopconditie in. Zorg dat het model stopt als elke turtle (minimaal) drie ontmoetingen met een andere turtle heeft gehad.



FIGUUR 2.40

```
ask turtles-here [  
  set ontmoeting ontmoeting + 1  
]
```

FIGUUR 2.41

Opdracht 22: Een mop doorvertellen II

In deze opdracht gaan we nogmaals een mop doorvertellen, maar nu met een andere tactiek. We laten de verteller rondlopen met een *random walk*. Hij kan zijn mop alleen doorvertellen als er een mens op dezelfde plek is die de mop nog niet kent. We gebruiken hiervoor net als in de vorige opgave -here. Als de blauwe verteller de mop heeft doorverteld wordt hij groen en is de ontvanger de nieuwe verteller.

158. Open het bestand *H2opg22.nlogo* en voer het model enkele keren uit.
159. In figuur 2.42 zie je de *go*-procedure. Beschrijf de betekenis van onderstaande coderegels in een Nederlandse zin:
`ifelse any? other mensen-here with [not kentMop?]`
Wanneer is dit waar en wanneer is dit niet waar?
160. Beschrijf de stopconditie van dit model in je eigen woorden.
161. In figuur 2.43 vindt op de met een pijl gemarkeerde patch een ontmoeting plaats. Door de kleur van de patch tijdelijk in roze te veranderen, wordt dit benadrukt, net als in de vorige opgave.
Leg uit hoe het komt dat de mens op de roze patch (nog) oranje gekleurd is, terwijl de blauwe verteller op dezelfde plek is als de oranje mens.
162. In de procedure `vertel_mop` is de coderegels `setxy pxcor pycor` uitgeschakeld met `;`. Schakel de regel in en zoek uit wat de exacte werking van deze coderegels is.

Het kost veel iteraties voordat alle mensen de mop gehoord hebben.

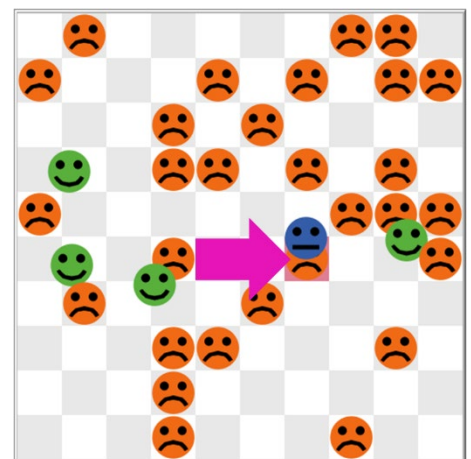
163. Verklaar dit.

We willen dat iedereen die de mop gehoord heeft deze blijft doorvertellen aan andere mensen (die de mop nog niet kennen). We onderzoeken welke invloed dat heeft op het totaal aantal iteraties.

164. Wijzig het model, zodat aan bovenstaande eis wordt voldaan.
165. Hoelang (hoeveel iteraties) duurt het om bij een bevolkingsdichtheid van 50% de mop door te vertellen?
LET OP: *one run is no run*. Run het model enkele keren uit om vast te stellen of er veel variatie in de uitkomst zit.

```
to go
  if not any? mensen with [not
    kentMop?] [
    stop
  ]
  ask mensen with [verteller?] [
    ifelse any? other mensen-here
      with [not kentMop?] [
        vertel_mop
      ]
    [
      loop_rond
    ]
  ]
  ask patches [
    stelOntmoetingVast
  ]
  tick
end
```

FIGUUR 2.42



FIGUUR 2.43

Opdracht 23: Aquarium V | etende vissen

We hebben in deze paragraaf technieken gezien waarbij agents elkaar ontmoeten en beïnvloeden. In ons aquarium gebruiken we dit als volgt: een vis die een alg ontmoet, eet deze op en krijgt daarvan energie. De alg verdwijnt doordat hij wordt opgegeten. (De kleur van de patch verandert hierbij van groen naar blauw.)

166. Open het bestand *H2opg18_AQ5.nlogo*, klik op *Setup* en vervolgens op *Go*.
167. Bestudeer de code en zoek uit hoe geprogrammeerd is dat de vissen algen eten.

De vissen krijgen nu steeds meer energie, want er is meer dan voldoende voedsel. Als vissen voldoende energie hebben, kunnen ze zich voortplanten en dat kost energie. Hiervoor is de procedure `plant_voort`.

168. Bestudeer procedure `plant_voort`. Die bevat de voor ons nieuwe functie `hatch`. Wat betekent dit?
169. De procedure `plant_voort` wordt op dit moment nog niet aangeroepen, want er staat een `;` voor. Haal de `;` weg, zodat de procedure uitgevoerd wordt en bekijk de uitvoer van het model.
170. De procedure `plant_voort` bevat meerdere `set`-opdrachten. Eén daarvan staat niet binnen maar buiten de blokhaken `[]` van de `hatch`-functie. Leg uit waarom dit de juiste keuze is.

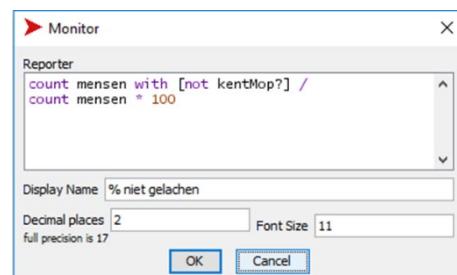
2.7 Globale variabelen en statistiek

Patches en turtles hebben **attributen**: specifieke eigenschappen gekoppeld aan individuele patches of turtles. Daarnaast hebben we in onze modellen gebruik gemaakt van sliders om **parameters** aan de modellen mee te geven, zoals *energie_uit_algen* in het aquariummodel en *bevolkingsdichtheid* bij het doorvertellen van moppen.

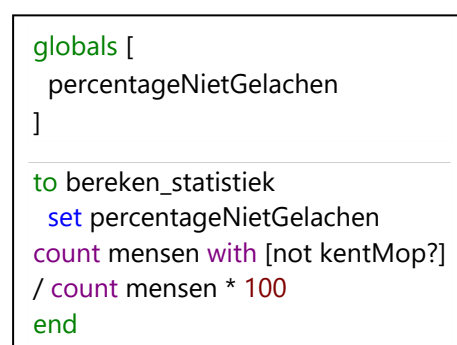
Parameters zijn instelbaar en dus variabel, maar meestal doe je dat alleen voordat je de *setup* uitvoert, want als je een parameter verandert tijdens het runnen van een model, beïnvloedt dit het model. Een parameter is juist niet gekoppeld aan één agent, maar (mogelijk) wel aan een groep patches, turtles of breeds. Je kunt de naam van de parameter overal in de code gebruiken. Een variabele met dat kenmerk heet een **globale variabele**.

In de opdrachten hebben we enkele keren gebruik gemaakt van monitors, zoals *% algen* in het aquariummodel of *% niet gelachen* bij het doorvertellen van moppen. Deze horen niet bij individuele agents maar bij het hele systeem (modelwereld). De bijbehorende waarden worden voor elke iteratie (tick) opnieuw berekend met een formule zoals bijvoorbeeld in figuur 2.44 voor *% niet gelachen*. De monitoren vertellen ons iets over de toestand van de hele modelwereld. Het is daarom logisch om hiervoor een globale variabele te gebruiken.

Een globale variabele maak je met `globals` (figuur 2.45). Zorg dat je dat helemaal bovenaan het model doet! Vervolgens kun je de variabele in een procedure gebruiken of aanpassen met `set`, zoals in de procedure `bereken_statistiek` in figuur 2.45. Merk op dat voor het berekenen van `percentageNietGelachen` exact dezelfde code is gebruikt als voor de *Reporter* van de monitor in figuur 2.44.



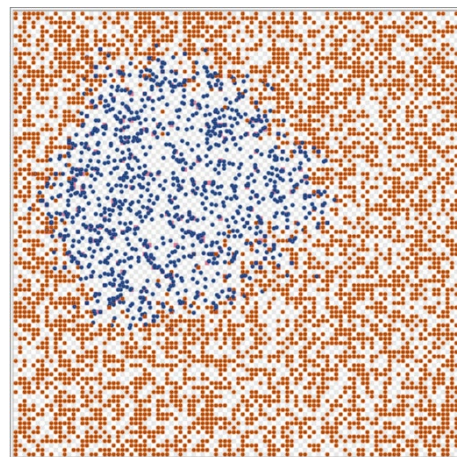
FIGUUR 2.45



FIGUUR 2.44

Opdracht 24: Een mop doorvertellen III

Open het bestand *H2opg24.nlogo*. Dit is het eindresultaat van opdracht 22, maar nu voor een wereld met veel meer patches (figuur 2.46). Bovendien is de globale variabele uit de theorie van deze paragraaf al gemaakt. Voer het model een keertje uit.



FIGUUR 2.46

171. De monitor *% niet gelachen* werkt nu nog met de *Reporter*-code in figuur 2.44. Kopieer deze code naar de procedure `bereken_statistiek`. Vervang de *Reporter*-code door `percentageNietGelachen` en voer het model nogmaals uit.

172. Leg uit waarom het verstandig is om de procedure `bereken_statistiek` al aan het eind van de *setup* aan te roepen.

173. Er zijn steeds meer (blauwe) vertellers. Hoeveel zijn het er precies?

174. Maak een nieuwe globale variabele `actieveVertellers`.

175. Breid de procedure `bereken_statistiek` uit, zodat deze de globale variabele `actieveVertellers` vult met het actuele aantal vertellers in onze modelwereld.

176. Voeg een monitor *aantal vertellers* toe die, gebruik makend van de globale variabele, laat zien hoeveel vertellers (blauwe agents) er op een bepaald moment zijn.

Het is lastig om gevoel te krijgen voor het aantal actieve vertellers. Immers: als we de bevolkingsdichtheid aanpassen, verandert ook het aantal mensen. Beter is het om een percentage actieve vertellers te maken.

177. Pas het model aan zodat deze werkt met een globale variabele `percentageVertellers` die steeds wordt herberekend in de procedure `bereken_statistiek` en wordt getoond in een bijbehorende monitor.

Opdracht 25: Een mop doorvertellen IV



Aan het eind van opdracht 22 hebben we ervoor gekozen dat elke agent die de mop gehoord heeft die mop blijft doorvertellen (en dus blauw blijft). Had je door dat het percentageVertellers uit opdracht 24 eigenlijk hetzelfde was als 100% - percentageNietGelachen?

Dat gaat nu veranderen. In een eerste versie van het model was het zo dat een verteller die de mop had doorverteld zelf groen werd (en stopte met doorvertellen). Dit is net zo min realistisch als dat iemand die een mop heeft gehoord die eeuwig blijft doorvertellen aan iedereen die hij tegenkomt. Zou het zo kunnen zijn dat iemand die een nieuwe mop heeft gehoord dat een beperkt aantal keren doorverteld en daarna stopt?

178. Open het bestand *H2opg25.nlogo*. Dit is het eindresultaat van de vorige opdracht.

179. Maak een nieuwe slider *maxDoorvertellen* (minimaal 1, maximaal 10, standaard 2) die aangeeft hoe vaak een mens een gehoorde mop maximaal mag doorvertellen aan een andere mens.

Om het model met de nieuwe parameter *maxDoorvertellen* te laten functioneren, moeten we nog een aantal aanpassingen doen:

- o Alle mensen moeten een attribuut *aantalDoorverteld* krijgen, zodat per agent kan worden bijgehouden hoe vaak deze een mop heeft doorverteld.
- o Bij de start van het model moet dit aantal voor iedereen op 0 staan.
- o Als iemand een mop heeft doorverteld, moet voor die agent *aantalDoorverteld* met 1 worden verhoogd.
LET OP: het maakt niet uit of hij de mop op dat moment aan één of meer mensen tegelijk vertelt.
- o Als een agent de mop net zo vaak heeft doorverteld als de met de parameter *maxDoorvertellen* ingestelde waarde, dan stopt hij met vertellen en krijgt hij een groene kleur.

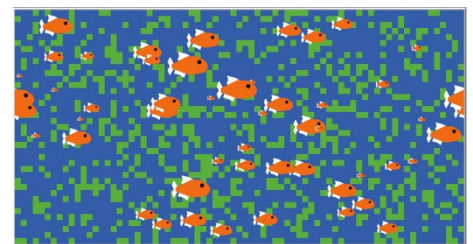
180. Pas het model aan volgens bovenstaande eisen.

181. Maak een monitor die laat zien hoeveel procent van de agents de mop wel heeft doorverteld maar op dit moment geen verteller meer is. (Er zijn meerdere manieren om de berekening te maken!)

Opdracht 26: Aquarium VI | groeiende vissen

Hoe kunnen we ons aquarium-model realistischer maken? Bijvoorbeeld met de volgende modeleisen:

- Als vissen ouder worden en voldoende energie hebben, groeien ze tot een maximale grootte.
- Hoe groter een vis is, hoe meer zwemenergie het kost om te zwemmen.
- Hoe meer algen er zijn, des te moeilijker deze groeien.
- Als er een bepaald percentage algen is, groeien ze niet verder (door gebrek aan licht en ruimte).



FIGUUR 2.47

182. Open het bestand *H2opg26_AQ6.nlogo*. Hierin zijn bovenstaande eisen geïmplementeerd.

Bovendien is een grafiek toegevoegd die het aantal vissen als functie van de tijd geeft. Hierover leer je meer in hoofdstuk 3. Klik op *Setup* en vervolgens op *Go*.

183. Bestudeer de code en onderzoek hoe is de eerste eis geprogrammeerd.

- o Aan welke eis moet worden voldaan om te kunnen groeien?
- o Wat is de maximale grootte van de vissen?

184. Hoe is geprogrammeerd dat een grotere vis meer zwemenergie gebruikt (eis ii)?

185. Op een bepaald moment is het percentage algen 40%. Wat is dan de kans dat een alg (met een blauwe patch naast zich) groeit (eis iii)?

186. Volgens eis iv groeien de algen bij een zeker percentage algen niet meer verder. Bij welk percentage is dat?

187. Maak een *slider* die een parameter *maxAlgen* instelt (minimaal 20, maximaal 50, standaard 40) die bepaalt bij welk percentage algen de algen niet meer groeien. Controleer de werking.

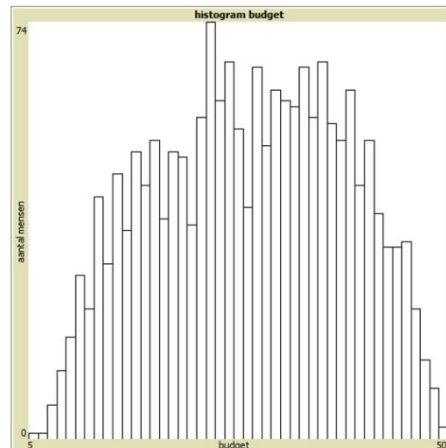
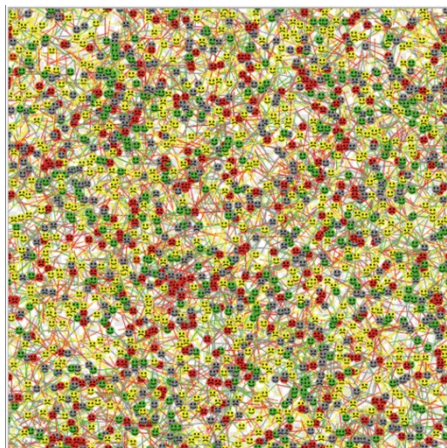
Opdracht 27: Samen delen | attributen vergelijken



In deze opdracht kijken we naar de casus *Samen delen*:

In een wereld leven mensen die in eerste instantie een random hoeveelheid geld krijgen. Dat is hun budget. Ze ontmoeten (door een *random walk*) andere mensen.

Als een agent iemand ontmoet met minder geld dan hij zelf heeft, dan geeft hij één eenheid geld van zijn eigen budget aan de ander.



FIGUUR 2.48

Hoe ontwikkelt zich dat?

188. Open het bestand *H2opg27.nlogo*. Dit model bevat een histogram (figuur 2.48 rechts) dat aangeeft hoeveel mensen er zijn met een bepaald budget. Klik op *setup* en daarna enkele keren op *1x Go*.
189. In het model kunnen mensen vier verschillende kleuren aannemen. Stel op basis van de code (en de simulatie) vast onder welke voorwaarden de agents een bepaalde kleur krijgen.
190. Leg voor zowel gele als grijze agents uit waarom ze blij kunnen kijken, maar ook verdrietig.
191. Maak een globale variabele `percentageBlij` waarin je bijhoudt hoeveel procent van de mensen blij kijkt.
192. Zorg dat de actuele waarde van `percentageBlij` in een monitor wordt getoond.

De procedure `wisselUit` (zie figuur 2.49; de regels met `shape` en `color` zijn hier weggelaten) laat zien hoe de ene mens zijn eigen budget kan vergelijken met het budget van de ander. De code vraagt enige uitleg:

Stel dat Alice rondloopt en Bob ontmoet. De opdracht `wisselUit` wordt dan gegeven aan Alice. Zij vraagt hierin aan Bob (hij is nu de `other` mensen-here):

(Alice:) *“Bob, wil jij jouw budget vergelijken met het budget van mij?”*

Deze vraag stel je met: `ifelse budget > [budget] of myself []`

Hierin slaat `budget` op het budget van Bob en `[budget] of myself []` op het budget van Alice (want zij stelde de vraag aan Bob).

Als de uitkomst van deze `ifelse` waar (`true`) is, dan heeft Bob meer geld dan Alice. In dat geval neemt zijn eigen budget met 1 af (omdat hij één eenheid geld aan Alice geeft). De regel:

```
set budget budget - 1
```

slaat dus op het budget van Bob.

De `myself` in deze code is nog steeds Alice die zichzelf tot slot vraagt om haar eigen budget op te hogen (met het geld van Bob):

```
ask myself [
```

```
  set budget budget + 1
```

```
]
```

193. Kies het juiste antwoord: een agent is blij als hij

A geld krijgt van een ander; **B** meer geld heeft dan de ander; **C** geld geeft aan de ander

194. Wat gebeurt er als twee agents met hetzelfde budget elkaar ontmoeten?

195. Vergroot de modelwereld naar 50×50 patches (met patchgrootte 10) en bekijk het eindresultaat.

```
to wisselUit
  ask other mensen-here [
    ifelse budget > [budget] of
  myself [
    set budget budget - 1
    ask myself [
      set budget budget + 1
    ]
  ]
  [
    set budget budget + 1
    ask myself [
      set budget budget - 1
    ]
  ]
  set label budget
]
end
```

FIGUUR 2.49

2.8 Eindopdrachten

In dit hoofdstuk heb je geleerd om zelf een model te maken volgens het principe van *agent-based modeling* met NetLogo. Met de opgedane kennis kun je nu zelf een model maken of een bestaand model verder uitbreiden.

De opdrachten hieronder geven suggesties voor verdere uitbreiding van de twee modellen waar we het meest mee hebben gewerkt: *aquarium* en *een mop doorvertellen*. Natuurlijk zijn er veel meer mogelijkheden: verzin zelf iets! Het is het leukst als je zelf een uitbreiding bedenkt en die implementeert.

Bij onderstaande suggesties zijn geen uitwerkingen beschikbaar.

Opdracht 28: Uitbreidingen | aquarium

Bij het uitbreiden van het model *aquarium* kun je denken aan:

- Vissen gaan niet alleen dood als ze te weinig energie hebben, maar ook als ze een maximale leeftijd bereiken.
- Vissen kunnen zich nu voortplanten bij voldoende energie. Als extra voorwaarde bouw je in dat er minimaal één keer een ontmoeting met een andere vis moet zijn geweest.
- Uitbreiding op de vorige suggestie: maak vissen met twee geslachten. Pas als een mannetje een vrouwtje heeft ontmoet kan er voortplanting plaatsvinden.
- Het tempo waarmee algen groeien hangt nu af van het totale percentage algen in het aquarium. Zorg dat de kans voor een alg om te groeien afhangt van de lokale situatie (lees: het aantal blauwe patches in de buurt): hoe meer blauwe patches, hoe groter de kans om te groeien.
- Introduceer een nieuwe soort giftige alg: bedenk zelf hoe deze invloed zal hebben op de gezondheid van de vissen.
- Vissen hebben algen nodig, maar ook zuurstof. Die halen ze uit blauwe patches. Geef vissen een zuurstofgehalte als eigenschap. Als je zwemt neemt het zuurstofgehalte af. Als je een blauwe patch ontmoet komt er zuurstof bij, maar bij een groene patch niet. Als het zuurstofgehalte 0 is gaat de vis dood.
- Variant of uitbreiding op de vorige suggestie: bouw in dat de vissen een minimaal aantal blauwe patches om zich heen moeten hebben om te kunnen ademen. Is de algen dichtheid te groot, dan sterven de vissen.
- Introduceer een nieuwe vissoort die geen algen maar andere vissen eet (bij een ontmoeting).
- Uitbreiding op de vorige suggestie: de vissen die andere vissen eten kunnen dat alleen doen, wanneer ze voldoende groter zijn dan de vissen die ze eten.

Opdracht 29: Uitbreidingen | een mop doorvertellen

Bij het uitbreiden van het model *een mop doorvertellen* kun je denken aan:

- Niet iedereen is een even goed verteller. Maak een eigenschap vertelkracht. Alleen als deze boven een bepaalde grenswaarde zit, wordt degene aan wie de mop wordt verteld gestimuleerd om zelf de mop door te vertellen.
- Uitbreiding op de vorige suggestie: hoe vaker je een mop vertelt, des te minder enthousiast je er zelf van wordt. Zorg dat de vertelkracht afneemt als iemand een mop vertelt.
- Uitbreiding op de vorige suggestie: niet iedereen is even ontvankelijk voor een mop. De een moet meer overtuigd worden om de mop door te vertellen dan de ander. Maak een eigenschap overtuigingskracht. Alleen als de vertelkracht van de verteller groter is dan de overtuigingskracht van de ontvanger, zal de ontvanger daarna zelf de mop doorvertellen.
- Je moet moeite doen om een mop door te vertellen, omdat je jezelf hiervoor moet verplaatsen. Op dit moment zoekt de verteller altijd door tot hij iemand gevonden heeft die de mop niet kent. Pas het model aan, zodat een verteller een maximum aantal pogingen doet (stappen verzet) om een mop door te vertellen.
- Een combinatie van bovenstaande suggesties: hoe meer je stappen je hebt gezet, hoe minder energie je hebt. Hierdoor wordt je vertelkracht ook kleiner.

BRONNEN

Algemeen

Voor deze module is gebruikt van het programma NetLogo, door Uri Wilensky. Het programma is te downloaden via: <https://ccl.northwestern.edu/NetLogo>

Veel afbeeldingen zijn screenshots uit dit programma. Het copyright van de gebruikte code is, waar dit relevant is, weergegeven in het meegeleverde NetLogo-bestand.

Bij opdracht 15 is gebruik gemaakt van de broncode van <https://playgameoflife.com>, ontwikkeld en beschikbaar gesteld door Edwin Martin.

Het SLO heeft zijn uiterste best gedaan om de auteursrechten te regelen van alle in deze module gebruikte informatie, zoals teksten, afbeeldingen en videofragmenten, en heeft toestemming gevraagd voor gebruik van dit materiaal voor deze module. Een ieder die zich niettemin eigenaar weet van dergelijk materiaal zonder dat direct of indirect met hem of haar afspraken zijn gemaakt, verzoeken wij contact op te nemen met het SLO.

Bronnen hoofdstuk 1

1.1	CCO	KNMI	https://www.knmi.nl/kennis-en-datacentrum/achtergrond/ijzel-begin-januari-2016
1.2	public domain	Stephen Ausmus	https://commons.wikimedia.org/wiki/File:Fire_ants_01.jpg
1.3	fair use	Endemol	https://en.wikipedia.org/wiki/File:Golden_Balls.jpg
1.4	-		
1.5	-		
1.6	permission	Tony Armstrong	https://secure.flickr.com/photos/tonyarmstrong/5410661753/sizes/l
1.7	-		
1.8	-		
1.9	-		
1.10	permission	René van der Veen	
1.11	permission	Fritz Vollrath & Richard Dawkins	Climbing mount improbable ISBN 0-14-017918-6
1.12	permission	Fritz Vollrath & Richard Dawkins	Climbing mount improbable ISBN 0-14-017918-6
1.13	CC BY-SA 4.0	Andrew Shiva	https://commons.wikimedia.org/wiki/File:RUS-2016-Aerial-SPB-Krestovsky_Stadium_01.jpg
1.14	-		
1.15	CC BY-SA 4.0	Michael Vadon & Gage Skidmore	https://commons.wikimedia.org/wiki/File:Trump_%26_Clinton.jpg
1.16	-		
1.17	-		
1.18	-		
1.19	-		
1.20	-		
1.21	-		
1.22	CC BY-SA 2.0	Naparazzi	https://www.flickr.com/photos/naparazzi/2984375333
1.23	-		
1.24		AvroTros EenVandaag	https://eenvandaag.avrotros.nl/item/de-voorspelbare-mens-2-biq-brother-en-criminaliteit-1
1.25	-		
1.26	-		
1.27	-		
1.28	-		

Bronnen hoofdstuk 2 & hoofdstuk 3

2.6	public domain	Lucas V. Barbosa	https://commons.wikimedia.org/wiki/File:Blue-cylinder.png https://commons.wikimedia.org/wiki/File:Blue-torus.png
1.3	fair use	Endemol	https://en.wikipedia.org/wiki/File:Golden_Balls.jpg
1.4	-		

INDEX

abstractie, 13
agents, 4, 5
analyseren, 15
Aquarium, 22, 30, 35
attributen, 18, 23, 33
attribuut, 19
boolean, 30
breeds, 26
button, 21
cellulaire automaat, 11
clear-all, 19
Color Swatches, 19
conceptualiseren, 13
count neighbors, 27
create-turtles, 23
definiëren, 13
doel, 13
eigenschap, 19
eigenschappen, 13
else, 22
emergent gedrag, 4
even, 28
experimenteren, 15
false, 30
formaliseren, 15
globale variabele, 33
go, 23
groepsgedrag, 4
hatch, 32
-here, 32
hypothese, 13
if, 22
ifelse, 22
initialisatie, 19, 23
iteratie, 6, 13, 23
keuze, 22
logische operatoren, 20
macrovalidatie, 15
microvalidatie, 15
mod, 28
model, 4
Model Settings, 18
modelleercyclus, 12, 13
monitor, 25
Netlogo, 17
niet-deterministisch, 9
omgeving, 4, 6
onderzoeksvraag, 13
one-of, 29, 30, 31
oneven, 28
own, 26, 30
parameter, 21
parameters, 33
patches, 18
procedure, 19
random, 9, 22
random-xcor, 23
reageren, 4
reflectie, 16
regels, 4
slider, 17, 21
sliders, 33
soorten, 26
sprout, 30
stopconditie, 25
ticks, 23
tijd, 6
toestand, 6
toeval, 8, 9
true, 30
validatie, 15
verificatie, 15
voorwaarde, 22
with, 27
wraparound, 19