Scratch (9)

René Suiker

In de crisis

We zitten ook in ons land midden in de coronacrisis en hebben veel doden te betreuren. En als CompUsers-leden behoren we toch voor het merendeel tot de risicogroep. Blijf vooral binnen en probeer je daar te vermaken. Het is wat wrang en ik wil er niet veel woorden aan besteden, maar met Scratch kun je jezelf binnen vermaken, al lokt de buitenwereld.

De vorige keer heb ik een aantal opgaven opgesteld, waar we nu mee gaan beginnen. We begonnen met het programma waarbij de mieren de taart opeten. Weet u het nog: https://scratch.mit.edu/projects/356124575/

Opgave 8.1: laat het programma stoppen als de hele taart op is.

We hebben nog niet voldoende stof behandeld om dit letterlijk te kunnen doen als de taart op is. We hebben nog geen middelen om te zien of er nog 'taart' over is op het veld. Wat je natuurlijk kunt doen om dit te benaderen, is een teller te introduceren, die het aantal hapjes telt. En als dit aantal hapjes overeenkomt met het aantal hapjes dat je verwacht dat er te eten valt, dan kun je het programma stoppen. En hoeveel hapjes zijn dat dan? Dat kun je natuurlijk 'empirisch bepalen'. Oftewel, in 'jip-en-janneketaal': je laat het programma lopen met de teller in beeld en als je vindt dat de taart op is, dan druk je op 'stop' en lees je de teller af. Vervolgens kun je die waarde gebruiken. Ik zou 'm iets lager leggen, want als er hier en daar nog een kruimeltje ligt, beschouwen we de taart als 'op'. Zeker zolang de mieren nog ongericht rondlopen, kan het nog een eeuwigheid duren voordat bij toeval de laatste kruimel gevonden wordt. Je merkt ook, naarmate het programma vordert, dat de 'hapjesteller' steeds langzamer gaat lopen, want er wordt relatief meer gewandeld dan gegeten.

In https://scratch.mit.edu/projects/381555543 heb ik een teller in beeld en de teller ging richting de 8.000 voordat de taart echt op was. Dit lijkt me een wat onrealistisch getal en het brengt me tot de eerste opgave voor vandaag. Programmeren in de praktijk is doen en fouten opsporen en herstellen. Jullie gaan me dus even helpen met foutzoeken, in vaktermen 'debuggen': de 'bugs' eruit halen.

Opgave 9.1: Kijk of het tellen goed verloopt: zo ja, prima; zo nee, wat gaat er verkeerd?

Opgave 8.2: Definieer een nest en een taart en laat de mieren de taart opeten

Dit begint al op 'virtual reality' te lijken. Je kunt dit natuurlijk zo mooi maken als je wilt en ik had gehoopt wat inzendingen te zien waar men zich fraai had uitgeleefd. Dit geeft ook even wat tijd voor een klein zijstapje naar het programmeren van games. Ik heb er niet veel verstand van, maar ik heb twee zoons die in die tak van sport wel hun sporen hebben verdiend en ik begrijp dat er grofweg drie soorten functies bij het tot stand komen van een spel betrokken zijn:

- De spelontwerper
- De artiest
- De programmeur

Die hebben alle drie een rol. En als je dus thuis programmeur bent, en je wilt een simulatie of een spel maken, dan moet je alle drie de rollen vervullen, maar het is onwaarschijnlijk dat je de drie rollen in dezelfde mate beheerst. Daarom worden de meeste programma's dan ook door teams gemaakt.
De spelontwerper is degene die het idee bedenkt, de scenario's uitwerkt, in gametermen de levels bedenkt. Dit is dus iemand die hier zegt dat we een speelveld hebben, met een mierennest, een taart, een hoop mieren en misschien wat manieren om die mieren te besturen en welke hindernissen er zoal kunnen optreden.

De artiest is degene die aan de gang gaat om het nest, de taart, de mieren en de omgeving zo af te beelden dat het past bij de sfeer en de bedoeling van de situatie. Dat is dus niet altijd 'zo realistisch mogelijk' maar dat kán het wel zijn. Sommige programma's hebben nadrukkelijk fantasiescenario's, zoals 'Lord of the Rings'. Andere spellen zijn zo realistisch mogelijk, zoals de 'FIFA'-reeks.

De programmeur moet zorgen dat de scenario's zoals beschreven en 'getekend' feitelijk datgene gaan doen wat er bedoeld is. Als ik in deze cursus programma's wil laten maken, dan besef ik dat we geen artiesten hebben, althans, dat verwacht ik. Dus ik hecht niet zo heel veel waarde aan de vormgeving, het programma moet werken volgens de opgave (de opgave was dus de taak van de spelontwerper).
Deze opgave zoals hij nu gedefinieerd is, is in feite te beperkt, te onduidelijk, daar konden jullie niet mee uit de voeten. Maar op basis van hetzelfde project ga ik nu de opgave iets anders definiëren en dit leidt tot de volgende opgave.

Opgave 9.2: Werk onderstaande opdracht uit

Voor deze opgave zou de spelontwerper in feite moeten zeggen:

- Ergens links staat een taart van ongeveer 200 hapjes;
- Rechtsonder staat een mierenhoop;
- Er zijn tussen de 40 en de 50 mieren, start bij het nest;
- Deze lopen volgens min of meer natuurlijke beweging;
- Deze lopen enigszins in de richting van de taart;
- Als ze bij de taart zijn nemen ze een hapje;
- Als ze een hapje hebben genomen, lopen ze naar het nest;

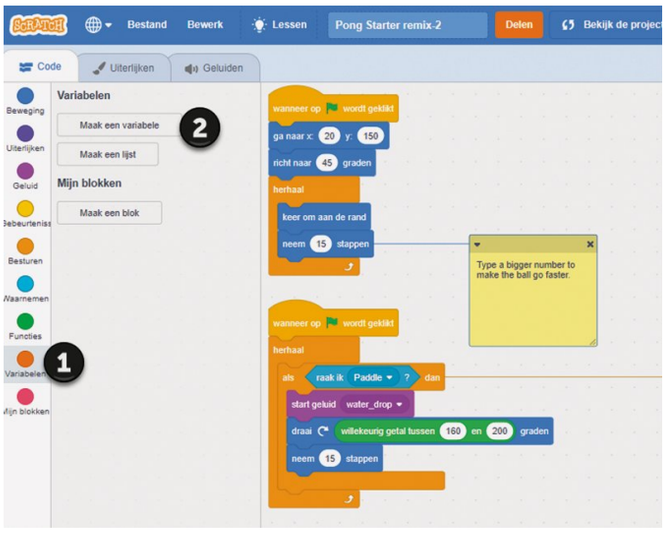
- Eenmaal bij het nest doen we alsof hap wordt gedumpt;
- Zolang er nog taart is, gaan we weer verder met stap 5;
- Je mag hierbij dus gebruik maken van de blokken van opgave 9.1.

Als je graag de echte artiesten-input wilt gebruiken: de mieren zijn al in het project aanwezig, je kunt natuurlijk een mierenhoop en een taart via Google (of een andere zoekmachine) opzoeken en gebruiken.

Dan de oude computertennis-variant voor de overige opgaven: https://scratch.mit.edu/projects/10128515/

Opgave 8.3: Voeg een scorebord toe, dat bijhoudt hoeveel keer het batje de bal heeft geraakt

Dit zou na alle uitleg niet zo moeilijk meer moeten zijn. De volgende stappen zijn nodig:
- Creëer een variabele 'score';
- Zet de waarde op 0 tijdens de initialisatie en toon deze;
- Verhoog de waarde elke keer dat de bal het batje raakt;
- Als we achter de schermen kijken, dan zien we twee sprites, te weten de bal (ball) en het batje (paddle).



Figuur 1 - Variabele

De code zit grotendeels bij de sprite 'ball', dus de variabele maken we ook hier aan. Bovendien willen we dat de variabele aangepast wordt als de bal het batje raakt, en het codeblok dat nu kijkt of ze elkaar raken hoort ook bij de bal. Dus is het logisch om daar de variabele te maken. Bij (1) zie je de selector voor variabelen, waardoor in het codeblok de commando's rondom de variabelen verschijnen. Bij (2) kunnen we de variabele maken. Wij noemen hem score, maar de naam van de variabele heeft voor de computer geen betekenis, het is alleen voor ons mensen handig om te onthouden waar de variabele voor bedoeld is. Als je hem liever een andere naam geeft, dat mag dus ook.
Als je klikt op 'maak een variabele' dan krijg je een 'pop-up' te zien zoals hieronder:

Figuur 2 - Variabele details

Bij (1) vul je dus de naam in, bij (2) kun je aangeven waar deze bij hoort. Je kunt dus ook bij één van de sprites een variabele maken die overal te gebruiken is. Dat doen we nu zelfs, want we hebben er geen last van en op het moment dat je hem aan één sprite koppelt, wordt het label op het scherm voorafgegaan door de naam van de sprite. Dus voor de werking hebben we het niet nodig, maar voor het label is het fraaier om hem voor alle sprites beschikbaar te houden.

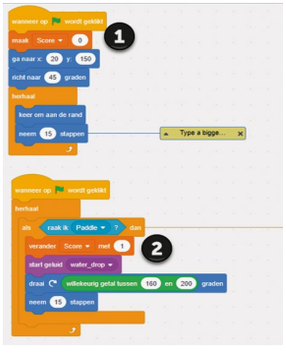
Je kunt zelfs een variabele in de cloud opslaan, ik kan me er wel iets bij voorstellen, maar ik weet nog niet precies wat hiervan de betekenis is. Mogelijk kun je dan dezelfde variabele delen met meerdere gebruikers, maar dat is in elk geval op dit moment nog niet aan de orde. Ik heb ook nog geen idee hoe dat dan zou moeten werken. Dus misschien iets voor de toekomst, maar misschien ook helemaal niet. En misschien wil iemand het alvast uitzoeken en er zelf een artikel over schrijven, dat mag natuurlijk altijd.

Als we de variabele gemaakt hebben, staat deze dus in het codeblok opgenomen en kunnen we hem in de code gebruiken.

Figuur 3 - Variabele beschikbaar

Je ziet de naam van de variabele genoteerd staan bij (1). Je ziet ook een vinkje voor de naam. Dat wil zeggen dat de waarde van de variabele op het speelveld getoond wordt. Dat willen we ook; het is de simpelste manier om de score in beeld te krijgen.

Op het speelveld kun je het scorebord gewoon schuiven naar waar je het hebben wilt. We hebben nu stap 1 gedaan en stap 2 gedeeltelijk. We moeten hem nog initialiseren, dat doen we op een plek die vrij snel na het starten aangeroepen wordt en buiten elke lus valt, want als je 'm binnen een lus op 0 zet, wordt hij elke keer weer op 0 gezet als de lus doorlopen wordt. En meestal worden lussen vaak doorlopen. In bovenstaand figuur zie je een opdracht 'Maak score 0' en een opdracht 'verander score met 1'. Het eerstgenoemde blokje zet je in de initialisatie, het tweede blokje voeg je toe bij de acties 'als de bal het batje raakt'. In de code ziet dat er aldus uit:



Figuur 4 - Codeblok score-initialisatie en bijhouden

Als je dit correct hebt uitgevoerd, dan zie dus de score in het speelveld en zul je merken dat de score correct wordt bijgehouden. Hiermee is opgave 8.3 tot een goed einde gebracht. Eenvoudig toch?

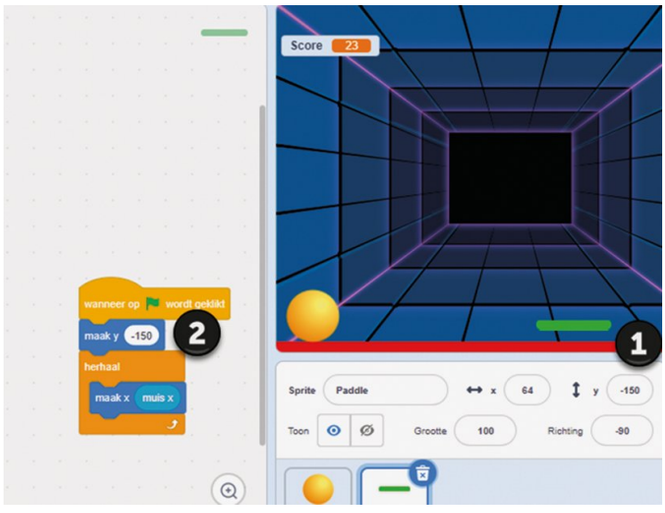
Opgave 8.4: Zorg dat het batje bij aanvang van het spel op de goede hoogte staat en bouw ook een vertraging in

Ook dit is in feite doodeenvoudig. Bij de vertraging gaat er natuurlijk om, dat je na het op de groene vlag drukken nog even tijd hebt om het batje te controleren. Zoals je in de code hebt kunnen zien volgt het batje de beweging van de muis in horizontale richting en blijft hij verticaal waar hij staat.

Je hebt hier wel twee verschillende stukjes code voor nodig. Het eerste deel van de vraag gaat namelijk over het batje en het tweede deel over de bal.

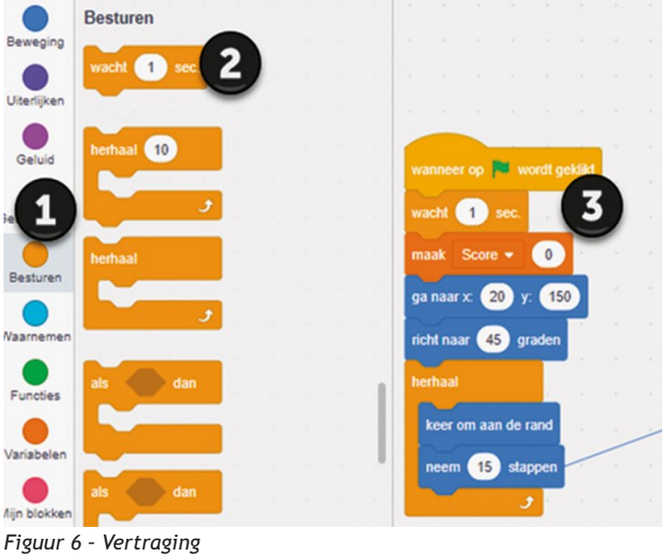
Dat het batje verticaal blijft waar het staat is prettig, je wilt niet dat hij alle kanten op beweegt, want dan wordt het spelletje wel heel simpel. Je kunt het natuurlijk wel eens proberen door 't zowel in x- als y-richting met de muis mee te laten bewegen, want tijdens programmeren is het altijd interessant om eens van de vraag af te wijken en iets zelf uit te zoeken.

Voor de startpositie gaan we als volgt te werk:



Figuur 5 - Startpositie batje

Je zet eerst het batje waar je het hebben wilt, net iets boven de bodem. Dan lees je bij (1) de juiste y-waarde af, in dit geval -150.
En uit het codeblok bij 'Beweging' pak je blokje (2) op en dat schuif je buiten de lus, want als je het blokje verder niet in verticale richting beweegt, hoef je deze waarde ook niet aan te passen. Overigens mag je 'm natuurlijk wel elke keer weer op -150 zetten, dus binnen de lus, want een 'grappenmaker' kan dan niet het batje oppakken en bovenin zetten. Ik weet ook niet of het kan tijdens het spel, maar zo kun je voorkomen dat het gebeurt. En, het is wel elke keer een extra instructie. Op deze schaal zal dat niet zo heel veel effect hebben, maar als je in je programma's instructies onnodig vaak uitvoert, wordt het geheel er niet vlotter van.
Onder de categorie 'besturing' bevindt zich de opdracht 'wacht 1 seconde'. Als je deze opdracht in de code voor de bal direct na de trigger van de groene vlag zet in het stukje dat de bal doet bewegen, dan valt de bal pas één seconde nadat je de groene vlag hebt aangeklikt Dit is wat we wilden. Je moet de vertraging natuurlijk niet in het blokje zetten dat kijkt of het batje geraakt wordt, want dan kan de bal er al doorheen gevallen zijn. In de code ziet het er als volgt uit:



Figuur 6 - Vertraging

U ziet: het huiswerk op dit vlak was niet onmogelijk.

Opgave 8.5: Maak de bal steeds iets kleiner als het batje wordt geraakt

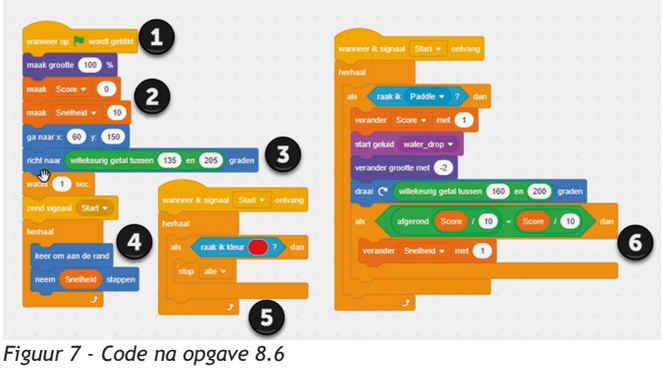
Dit moet natuurlijk helemaal geen probleem meer zijn. Op dit moment is de bal de hele tijd even groot. De sprite is opgezet met een bepaalde grootte en die blijft de hele tijd van kracht. Onder de categorie 'Uiterlijken' heb je de gelegenheid iets met de grootte van het object te doen.
Je kunt de opgave heel letterlijk nemen en echt elke keer dat het batje geraakt wordt, iets met de grootte doen, of je doet het subtieler, zeg maar elke vijf keer, maar het idee is hetzelfde. Als we de score aanpassen, dan passen we ook de grootte van de bal aan.

Opgave 8.6: Laat de bal ook sneller gaan, als het batje een veelvoud van tien keer is geraakt

Als je de code goed bekeken hebt, dan zie je dat de beweging als volgt verloopt:
- De bal wordt een richting gegeven;
- De bal zet 15 stappen;
- Als er niets bijzonders gebeurt, ga naar 2;
- Als het batje geraakt wordt;
- Ga in een andere richting en zet 15 stappen;
- Ga naar 2;
- Als de bodem geraakt wordt, stop alles.

Die snelheid wordt dus bepaald door de 15 stappen. Als we dit willen veranderen, moeten we dus die 15 vervangen door een variabele. Laten we die maar snelheid noemen.
Ik merkte trouwens dat het programma af en toe niet lekker begon. Terwijl ik de beginpositie van de bal echt bovenin had staan, bleef hij toch bij de start wel eens beneden liggen. Daar kun je lang naar zoeken als programmeur, maar ik kan wel aangeven waar de crux zat in dit geval. We hadden drie codeblokken die alle drie tegelijk startten. En het eerste blok zette de bal bovenin, maar tegelijkertijd keek de code al of de bal het batje of de bodem raakte. Dat is niet handig, want als het spel uit is, dan kan het wel eens allemaal het geval zijn. En zodoende zag ik dat het programma bij het opstarten wel de teller liet oplopen (die ging dus niet naar 0) maar wel gelijk stopte.

Om dit op te lossen, letten we niet voor alle blokjes op de groene vlag, maar gebruiken we de groene vlag alleen om de initialisatie te starten. Als die achter de rug is geven we een signaal 'start' af, waar de andere blokjes dan op mogen reageren.
Goed, om alles nu lekker te laten verlopen, de bal steeds kleiner te maken en de snelheid steeds hoger, hebben we de volgende code nodig. Kijk rustig of je alles kunt verklaren en kijk ook gelijk, of je het zelf misschien makkelijker of efficiënter kunt maken. Ik heb in elk geval met dit stukje code iemand enthousiast kunnen maken voor Scratch, want met zo weinig code het pingpongspel bouwen, dat zegt wel wat over de flexibiliteit en kracht van Scratch.



Figuur 7 - Code na opgave 8.6

Bij (1) zie je dus dat we de vlag als trigger voor de initialisatie gebruiken. De code bij (2) dient ter initialisatie. We zorgen dat de bal in uitgangspositie komt; dat wil zeggen: startpositie, normale grootte, startsnelheid is nu 10. Bij (3) richten we de bal min of meer naar beneden, iets vaker iets naar rechts dan iets naar links, maar daar mag je zelf mee spelen. Bij (4) wordt het normale verloop geregeld, zeg maar de beweging van de bal als er niets bijzonders gebeurt. Bij (5) wordt geregeld dat het spel stopt als je de bal de bodem laat raken. Bij 6 wordt ervoor gezorgd, dat de snelheid alleen maar verandert als de score verandert met een tienvoud, dus niet elke keer. Zo'n zelfde blokje heb je ook nodig bij de volgende opgave.

Opgave 8.7: Maak het batje kleiner, elke 25e keer dat de bal wordt geraakt

Je zou zeggen, een zelfde blokje als blokje 6 hierboven, en dan niet delen door tien maar door 25, dan kun je het batje kleiner maken. Maar er zit hier toch een complicerende factor bij. Die kan ik natuurlijk wel uitleggen en ook oplossen, maar ik ben toch benieuwd of iedereen intussen in zoverre mee is gegaan, dat jullie dit ook allemaal zien. Dus de vraag luidt:

Opgave 9.3: Wat is de complicerende factor bij opgave 8.7?

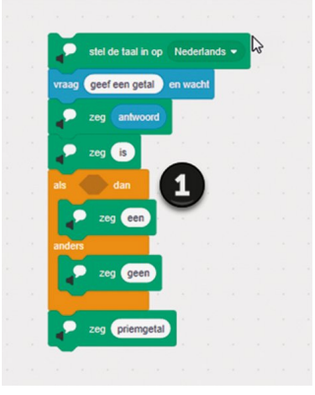
Opgave 8.8: Probeer een score van 100+ te halen

Veel plezier met het spel en met de opgaven. Als je alle opgaven hebt afgerond, dan zal je merken, dat het nog niet zo eenvoudig is om die score van 100 te halen. De bal en het batje worden steeds kleiner, terwijl de snelheid steeds verder toeneemt. Maar niets is onmogelijk, dus ik laat me graag verrassen door de hoogste scores.
Het is natuurlijk leuk, dat ik de vorige keer veel huiswerk heb opgegeven, maar dat betekent, dat ik binnen de beschikbare ruimte niet veel gelegenheid meer heb om nog nieuwe onderwerpen in te brengen. Nu hoeft dat in feite ook niet meer, want intussen hebben jullie zo veel geleerd en met name geleerd hoe je aan informatie moet komen, dat in feite de hele theorie wel afgerond is. We kunnen natuurlijk stilstaan bij alle commando's en uitbreidingen die nu beschikbaar zijn, maar ik denk niet dat dat nu zoveel toevoegt.

Ik denk dat het zinvoller is, om leuke uitdagingen te definiëren, die dan de volgende keer weer besproken gaan worden. Eén van de suggesties bij de 'pong starter', het spel dat we zojuist speelden, was om een extra bal in het spel te brengen. Op dit moment weet ik nog niet, hoe dat moet gaan verlopen, of het zeg maar mogelijk is om twee ballen in het spel te hebben en het ook nog speelbaar te houden, maar het is het proberen waard. Dus de volgende opgave voegen we nog toe aan het huiswerk:

Opgave 9.4: Voeg bij een score van 50 een extra bal toe

Houd hierbij rekening met het volgende:
Misschien moet de mate waarin de bal verkleind wordt iets vertraagd worden.
Als er een bal gemaakt wordt, lijkt die dan op de bal in het spel, of op de originele bal?
Als er een bal gemaakt wordt, welke snelheid krijgt die dan mee?
En ten slotte, nog één opgave te gaan. Vorige keer hadden we er acht, dat is wat veel van het goede. Maar vijf opgaven, dat moet toch te doen zijn. In principe verschijnt deel 10 in SoftwareBus nummer 6 van dit jaar, dus je hebt nog tijd om het uit te werken. We gaan wel eens een andere weg in. Ik laat je wat opdrachten zien, maar er is een stukje nog niet ingevuld. Ik denk dat jullie wel begrijpen wat je moet doen.



Figuur 8 - Opgave 9.5

Het is de bedoeling, dat jullie bij (1) iets invullen, waarvoor je eventueel meer code nodig hebt, zodat de code doet wat je verwacht.

Succes!