Scratch (15)

René Suiker

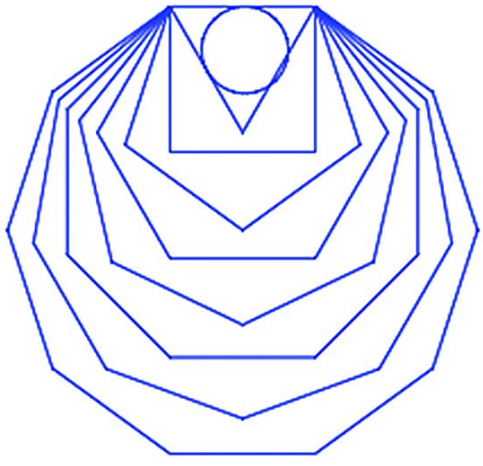
Ervan uitgaande dat we de ogen en vingers nog hebben, gaan we ook dit jaar weer op een laag pitje verder met Scratch. Niet omdat het moet, maar vooral omdat het gewoon heel leuk is. Het lage pitje komt vooral voort uit het feit dat ik bijna alles al behandeld heb, maar er is natuurlijk altijd wel iets te leren. En zo kwam het dat ik zowaar weer een idee kreeg, los van het uitwerken van het huiswerk.

Vaste lezers hebben intussen door dat ik heel enthousiast ben over Scratch, want het oogt zo simpel, maar je kunt er zo veel mee. Ik zag onlangs ook een voorbeeld van iemand die een 'bricks' spel maakte met Scratch en dat was heel eenvoudig. Bricks is een klassieker, je moet met een batje een balletje op een wand richten en elke keer als het balletje de want raakt breekt er een steentje. En de bedoeling is dat je alle stenen wegspeelt.
Uiteraard kun je het zo ingewikkeld maken als je wilt, maar hij nam de kijker mee in het proces, van heel eenvoudig tot erg gelikt. Voor wie de Engelse taal machtig is, deze meneer, Al Sweigart, heeft een boek over Scratch online geplaatst en dat kun je dus gratis lezen en dat behandelt eigenlijk alles van Scratch: **https://inventwithscratch.com**
De lessen die hij online zet volgen dat boek, dus als je het Bricks-spel zelf wilt maken kan ik je die site van harte aanbevelen. Maar ik wilde dus iets anders.

Zoals jullie weten ben ik ook met een ander product bezig, met de intentie daar ook een reeks verhalen over te schrijven en dat gaat over Unity. Unity is een zogenaamde 'Game engine', zie elders in dit nummer. En nu wordt Scratch af en toe als speelgoed gezien, terwijl Unity dus professioneel is. En ik moet toegeven, Unity bevat heel veel krachtige elementen die het de spelontwikkelaar makkelijker moet maken een spel te ontwikkelen, maar ik wil eigenlijk eens kijken of ik met Scratch ongeveer hetzelfde kan bereiken als met Unity. Uiteraard zal ik een keer tegen de beperkingen van Scratch aanlopen, maar ik kan alleen maar zeggen (en straks aantonen): ***je moet de mogelijkheden van Scratch niet onderschatten***.
Op vrijdagavonden volg ik een reeks van bijeenkomsten van de IG Programmeren, waarbij zij met Unity bezig zijn. Ik leer daar veel van en ze zijn onlangs weer begonnen om zich op beginners te richten, waartoe ik mijzelf ook reken. Zelfs op het gebied van Scratch kom ik net kijken, maar van Unity weet ik nog veel minder. Marco Kurvers, die de leiding heeft over deze bijeenkomsten, wilde mij wel bijstaan in de reeks van artikelen over Unity en ons idee was, om te beginnen met de mogelijkheden van Unity zonder te programmeren.
En hierbij het ik het idee, zie ook het andere artikel, om een deeltjesversneller te bouwen. Nu is dat in het echte leven iets wat je niet zomaar in elkaar zet, maar ik had ooit eens een film gezien, over een goedje dat stuiterde zonder dat er energie verloren ging, maar waarbij juist energie werd gewonnen. Elke keer dat het de grond raakte, ging het harder omhoog dan dat het viel. Nu was dat een film en daarin is alles mogelijk en ik denk ook niet dat deze stofjes bestaan, maar het idee is wel leuk en eenvoudig te simuleren. In Unity gingen we met dat concept aan de slag en ik kreeg het idee om hetzelfde concept in Scratch ook uit te proberen.

Huiswerk

Maar eerst blikken we nog even terug op het huiswerk. Het was het huiswerk van aflevering 14, maar dat greep weer terug op de veelhoek die we in aflevering 13 beschreven. Althans, de veelhoek, het waren er een aantal, in elkaar. Het resultaat moest er ongeveer zo uitzien:



Figuur 1 - Resultaat 'veelhoeken'

Ik had de code laten zien en er vielen een paar dingen op waar we niet heel gelukkig mee waren. Zie trouwens SoftwareBus 5 van 2021 voor de volledige uitwerking; voor abonnees is deze ook online volledig beschikbaar. Ik liet de uitwerking zien via een paar plaatjes:
1. De hoofdlus
2. De initialisatie
3. De veelhoekfunctie
4. De cirkel

En dan kijken we nu terug naar de vragen:

Opgave 14.1:
a. In welk blok zat de fout:
   a. In de hoofdlus
   b. In de initialisatie
   c. In de definitie van de veelhoek
   d. In de definitie van de cirkel
b. Gegeven dat het programma eerst de cirkel niet geheel tekende, maar ook veel trager draaide, heb je enig idee wat de fout geweest kan zijn?

Opgave 14.2:
a. De opdracht was om zo compact mogelijk te programmeren. Zie je nog een manier om het programma, met behoud van functionaliteit, compacter te maken?
b. Zie je een manier om het programma efficiënter te laten lopen, dus eventueel met meer code, maar snellere uitvoering?

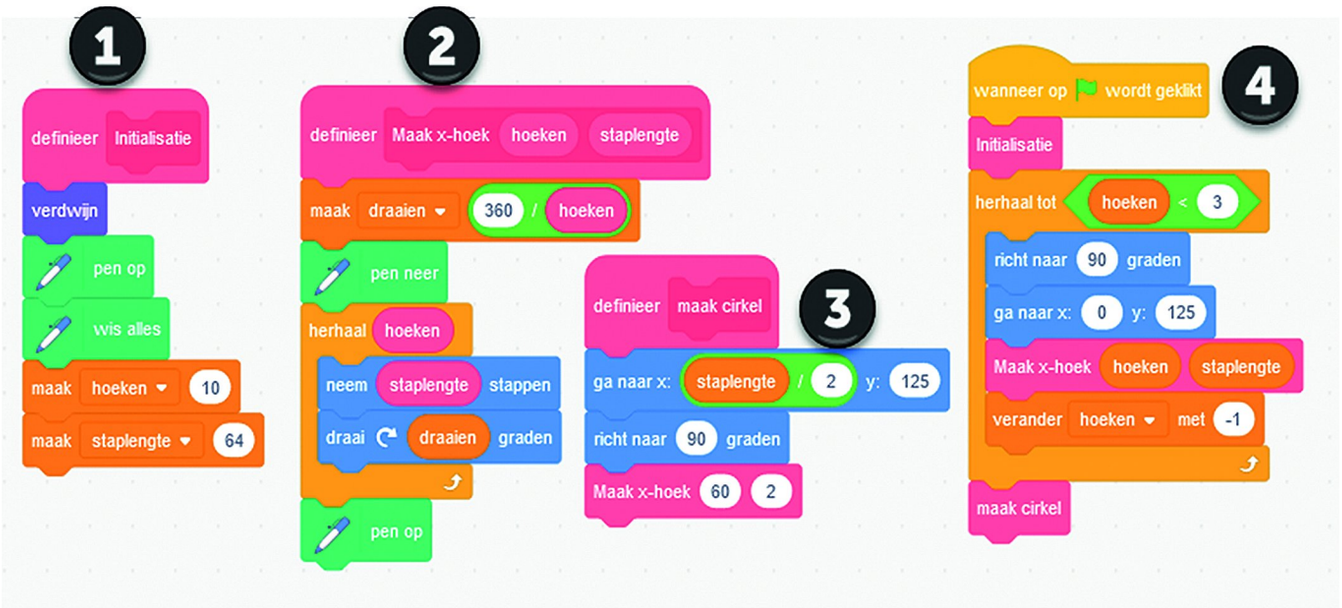
Opgave 14.3:
a. Pas het programma zo aan dat elke veelhoek een andere kleur krijgt
b. Pas het programma zo aan dat de cirkel rond het midden wordt getekend en dan de veelhoeken er omheen.

Uitwerking

Opgave 14.1:
De fout zat uiteraard in de veelhoekfunctie. Dat was ook de grootste kanshebber, want daar vond de complexere logica plaats. En met programmeren zitten fouten vaak in een klein hoekje, maar meestal wel rondom de complexere zaken.
We hadden te maken met een variabele 'draaien' en een variabele 'hoeken'. De variabele 'hoeken' bevatte een waarde tussen 3 en 10. De variabele 'draaien' gaf aan hoeveel graden per hoek gedraaid moest worden. En de herhaalfunctie

moest dus per hoek doorlopen worden, maar omdat hij werd aangeroepen met ‘draaien’ in plaats van ‘hoeken’ werd hij dus heel veel keer doorlopen, namelijk bijvoorbeeld 120 keer als het een driehoek betrof. Dus de driehoek werd wel getekend, alleen niet in drie streepjes, maar in 120 streepjes. Omdat ze over elkaar heen vielen, viel dat niet op.

De totale (aangepaste) code ziet er nu als volgt uit:



Figuur 2 - Totale code voor de veelhoek

Onder (**1**) zie je de initialisatie. Behoorlijk recht-toe-recht-aan. Onder (**2**) de gecorrigeerde versie van de functie ‘Maak x-hoek’. Onder (**3**) zie je de vernieuwde functie ‘Maak cirkel’ en bij (**4**) zie je de hoofdlus.

Al met al is de code al redelijk compact en worden de veelhoeken redelijk vlot getekend. De cirkel doet er iets langer over, maar dat komt omdat hier de code in de veelhoek 60 keer doorlopen wordt. Je ziet die cirkel dus stapje voor stapje opgebouwd worden. Overigens, als je de code in de turbo-mode laat uitvoeren, dan staat het eindfiguur er ogenblikkelijk.

Opgave 14.2:
Dit was een opgave voor de cracks, want zelf zie ik niet zo gauw veel verbetermogelijkheden. Maar ik ben dan ook geen professioneel programmeur. Misschien dat een van de lezers vanuit de IG Programmeren of IG Kunstmatige Intelligentie hier nog mogelijkheden ziet? Maar het kan ook net zo goed iemand anders zijn die ineens het licht ziet. Ik ben benieuwd.

Opgave 14.3:
Deel a is natuurlijk heel eenvoudig. Er is onder de groep ‘Pen’ (donkergroen) een functie beschikbaar ‘verander pen kleur met een waarde’. Als je dit toepast binnen de hoofdlus bereik je snel bijna het gewenste effect *(zie figuur 3 hiernaast)*

Toch doet dit net niet helemaal wat we beogen, al komt het aardig in de buurt. Dat brengt ons dus bij het nieuwe huiswerk:

Opgave 15.1:
a. Waarom is dit het niet helemaal?
b. Wat kun je daaraan doen?

Opgave 14.3.b was iets complexer; niet zozeer om het helemaal in beweging te zetten, want daar is de huidige code geschikt voor, maar je moet de coördinaten even goed uitpluizen.

We beginnen maar met de cirkel: die zetten we rond het midden. Dus we draaien de volgorde binnen de hoofdlus even om, eerst de cirkel en dan pas de veelhoeken. Het kan helpen om eerst een assenkruis te tekenen, zodat je een beeld hebt waar je moet zijn. Daarna is het óf wat gevorderde wiskunde, óf empirisch vaststellen waar elk figuur moet beginnen, zodat je met de veelhoek rondom de cirkel uitkomt. Als je een algoritme kunt vaststellen, dan kom je vast een eind.

Ik laat het nu even aan de lezer over, want ik wil iets nieuws gaan behandelen. Eventueel kan ik volgende keer nog wel iets dieper op deze materie ingaan, mocht dat aan de orde zijn.

Wel is het misschien handig om tijdens het ontwikkelen niet gelijk alle veelhoeken in één keer te tekenen, maar eerst bijvoorbeeld een driehoek. Kijk eens goed waar die moet komen en hoe zich dat tot het assenkruis en de cirkel verhoudt. Als je dat voor de driehoek hebt vastgesteld, lukt het met het vierkant ook wel en zo ga je verder. De grootte van het object hangt op grond van de logica in de functie wel af van het aantal hoeken. Dus moet je ook je startpositie hiervan af laten hangen.

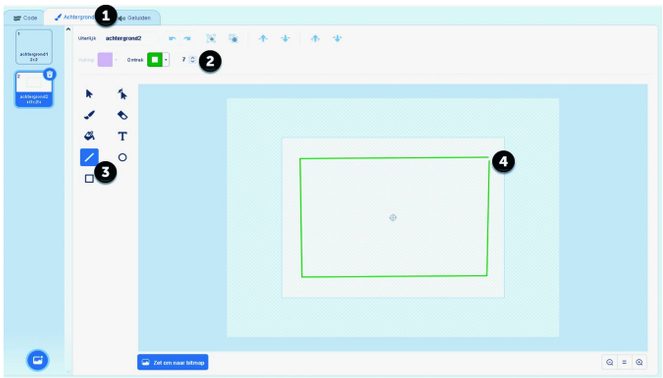
Figuur 3 - Opgave 14.3.a >

Opgave 15.2:
a. Probeer de hiervoor genoemde opgave 14.3.b nu zelf, met de gegeven aanwijzingen
b. Welke handige tips kun je delen om te slagen?

En toen iets nieuws

Zoals ik hiervoor al aangaf wil ik een rudimentaire deeltjesversneller bouwen. Niet zozeer omdat ik er een toepassing voor heb, maar als oud-marineofficier heb ik iets met explosies en de wetten van de natuurkunde en het is natuurlijk leuk om een simulatie te maken die daarmee kan spotten. Nogmaals, een echte deeltjesversneller is natuurlijk heel complex en het voert veel te ver om zelf maar het minieme beetje dat ik er wel van snap uit de doeken te doen, laat staan dit proces helemaal te beschrijven. Maar het idee ontstond toen we met Unity bezig waren. Unity is dus geen programmeertaal, zoals Scratch wel is, maar is een zogenaamde engine, die dus het programmeerwerk voor een deel overneemt. En om een goed spel te maken, moet je dus programmeren én kun je Unity inzetten om een deel van je werk te vergemakkelijken. Dit werkt nog niet met Scratch, want dan moet je C# gaan leren. Dat gaan we in de verhalen over Unity te zijner tijd wel wat uitwerken, maar voorlopig wilden we even binnen Unity blijven en daar wat mee stoeien, voordat we feitelijk gingen programmeren. En binnen Unity heb je wat fysische wetten ingebouwd waardoor je je er als programmeur niet al te druk over hoeft te maken. Dat iets valt komt door de zwaartekracht en die kun je in Unity instellen, waardoor objecten daar dus ook vallen, totdat ze op de grond liggen. En als die grond nogal schuin is, kunnen objecten er ook vanaf rollen. En wat heeft dit nu allemaal met Scratch te maken? Niets, helemaal niets. Behalve dat ik wilde zien in hoeverre het experiment dat we met Unity gingen doen ook met Scratch mogelijk is. En zo begon dus een tweetal artikelen die op elkaar ingrijpen. Want in dit Scratch-artikel kijk ik met een schuin oog ook naar Unity en in het Unity artikel verwijs ik ook naar Scratch.

Mijn conclusie is op voorhand, maar dat gaan we dus uitproberen, dat Unity krachtiger is, maar dat Scratch makkelijker is en dat je met beide tools iets boeiends kunt maken.
Het idee is om een soort box te maken waarbinnen een balletje heen en weer stuitert, maar het moet steeds harder gaan stuiteren. In die box zit een gat waar die bal dan een keer met hoge snelheid doorheen schiet.
In ons doolhofspel, enige tijd geleden, hebben we gezien hoe je iets kon laten bewegen en kon versnellen, dus die techniek kunnen we weer toepassen. Toen ging het met keyboardbesturing, nu moeten we het automatisch laten gebeuren. We kunnen dit zo mooi en realistisch mogelijk maken met invloeden van zwaartekracht e.d., maar laten we eerst maar eenvoudig beginnen. Het balletje valt naar beneden, stuitert omhoog tegen het plafond en stuitert dan weer terug, telkens bij elke stuit een beetje snelheid winnend. Het is de bedoeling dat hij niet alleen maar recht stuitert, maar een beetje rond, zodat hij ook een keer bij de uitgang komt. We beginnen maar even met de achtergrond:



Figuur 4 - De achtergrond

We maken de achtergrond door rechtsonder op achtergrond te klikken en dan een nieuwe aan te maken. Eenmaal geselecteerd zie je bij (**1**) dat het tabblad achtergronden geselecteerd is. Met behulp van ‘Omtrek’ bij (**2**) kun je de lijnkleur en -dikte bepalen. Ik maak gebruik van een kleine sprite (de tennisbal) en die laat ik in een relatief grote box stuiteren. De lijnen van de box zijn ook dik. Het is aan te bevelen de lijnen allemaal dezelfde kleur te geven, omdat we straks gaan kijken of die kleur geraakt wordt.
Bij (**3**) zie je het gereedschap om lijnen te tekenen. We zouden eronder ook een figuur als een rechthoek kunnen tekenen, maar dan moeten we er later een gat in maken als uitgang. Ik heb dus gewoon vier aaneengesloten lijnen gemaakt. Dan heb ik rechtsboven bij (**4**) een opening overgehouden, waar het balletje net doorheen zou kunnen.
We gooien dus de kat even weg als sprite en kiezen de tennisbal. Ik neem aan dat bekend is hoe dat moet. De tennisbal wordt gewoon meegeleverd bij Scratch, daar hoef je dus niet veel moeite voor te doen.
De code in deze simulatie wordt dus opgehangen aan de sprite-tennisbal. Om te beginnen moeten we het stuiteren goed ondervangen. Ik heb daarvoor het volgende blokje ergens in de code nodig:

Figuur 5 - Stuiteren

We hebben een blok nodig uit de (oranje) categorie ‘besturen’ voor een ‘als-dan’-constructie. Wat we ons afvragen uit de (lichtblauwe) categorie ‘waarnemen’ is of we een specifieke kleur aanraken. Als je op de kleur klikt kun je een kleur selecteren. Deze kleur selecteren we met het pipetje, we nemen de kleur van de rand van de achtergrond.
Als deze waarde optreedt, dan keren we de bal om. Hier zie je een rood blok, dat wil zeggen, een eigen blok. De code voor ‘keer om’ moeten we nog schrijven.
Om te kunnen omkeren moeten we een variabele snelheid en richting hebben. Scratch kent niet zoiets als een automatische beweging op basis van snelheid en richting, we kunnen echter wel de sprite een aantal stappen in een bepaalde richting laten zetten. Deze gaan we dus aanmaken onder de (bruine) categorie ‘Variabelen’. Deze kunnen we aanmaken voor deze sprite alleen of voor alle sprites. Dat maakt hier niet uit, omdat we maar één sprite hebben.
We maken dus een variabele ‘Snelheid’, want ‘Richting’ is binnen Scratch wel al bekend (onder de categorie ‘Beweging’). Deze laten we ook zichtbaar zijn op het speelveld. Standaard komen die onder elkaar, maar als je de box redelijk groot hebt, is het misschien logischer om ze naast elkaar te zetten. Het speelveld ziet er nu aldus uit:

Figuur 6 - Speelveld

Vervolgens maken we ook nog een eigen blok ‘Initialisatie’. Hierin zetten we de startwaarden, dan hoeven we die niet steeds in het codeblok van de hoofdlus op te zoeken als we hieraan willen sleutelen.
De hoofdlus wordt dan wel heel simpel:

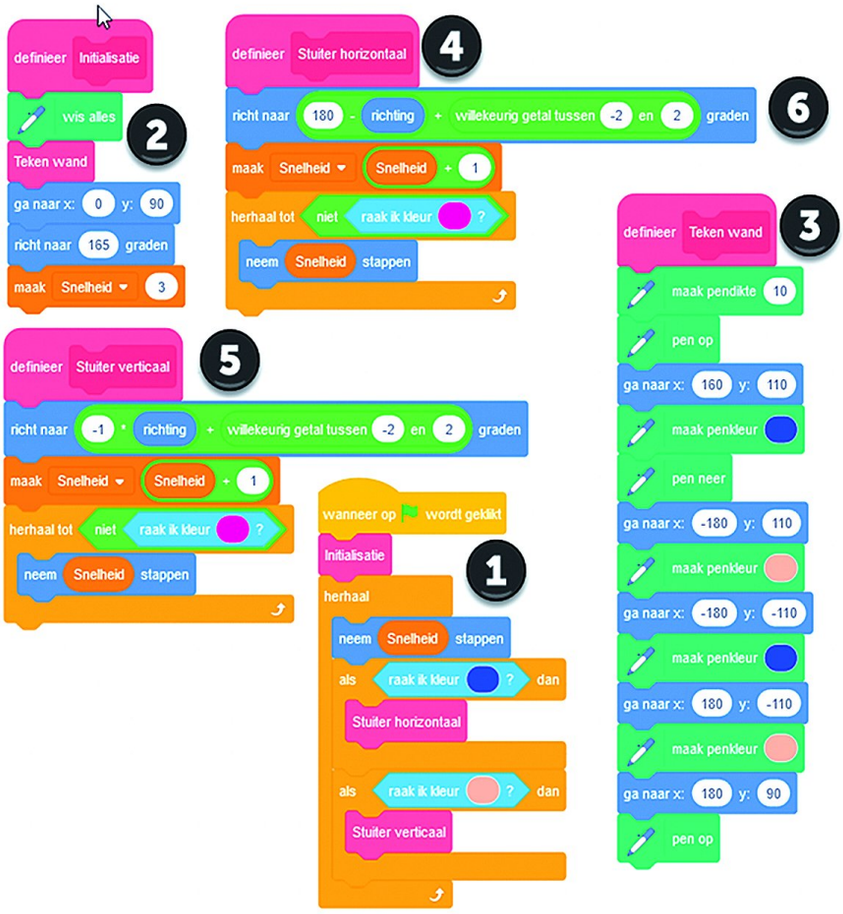
- Initialiseren
- Eeuwige lus:
  - Ga met snelheid ‘snelheid’ in de gekozen richting ‘richting’.
  - Als je een rand raakt, keer om.
- Dit gaat door tot je op de rode vlag drukt. Dan kun je de snelheid en richting uitlezen van je scherm en met andere startwaarden kijken wat er gebeurt.

De grote uitdaging zit hem in de juiste code voor het omkeren. Tijdens proefdraaien ging de bal telkens door de zijkanten van de rechthoek. Het komt blijkbaar zeer nauw, wat overigens ook het geval was bij Unity. Daar zijn we nog steeds aan het uitzoeken waarom de bal af en toe door de wand heen schiet, zeker bij hogere snelheden.
Dat komt waarschijnlijk doordat de programmatuur zich afvraagt of je iets raakt aan het eind van een beweging. Als de beweging heel snel is, is het object al door de grens heen voordat die opgemerkt wordt. Bij Scratch, waar je dit zelf moet programmeren, zie je dit gebeuren. Bij Unity, waar de engine zorgt draagt voor de mechanica, zie je dat niet direct.
Verder had ik bedacht dat, bij het stuiteren, de ‘richting van inval is de richting van uitval’ uitgevoerd kon worden door: Nieuwe richting = 180 – oude richting. Dit gaat echter alleen op voor het stuiteren op een horizontaal vlak. Voor stuiteren op een verticaal vlak geldt een andere formule. Voor een stuit tegen een verticale wand geldt dat de hoek met -1 vermenigvuldigd wordt. Verder is het ook leuk om niet exact te stuiteren, dus om de baan iets minder voorspelbaar te maken. We willen dus de berekende hoek een klein beetje laten variëren. De totale code ziet er nu als volgt uit:

Uiteindelijk is het allemaal dus toch ingewikkelder geworden dan ik me in eerste instantie had voorgesteld, maar het past nog steeds redelijk makkelijk op één scherm.
De hoofdlus onder (**1**) zegt in feite:

1. Initialisatie – verder beschreven bij (2)
2. Eeuwige lus:
   a. Neem ‘snelheid’ stappen
   b. Kijk of je een horizontale wand raakt
      i. Indien ja, stuiter horizontaal
   c. Kijk of je een verticale wand raakt
      i. Indien ja, stuiter verticaal

Voor alle zekerheid heb ik geen achtergrond getekend, maar een box, waardoor ik volledige controle had over de posities en de opening. Je ziet in de hoofdlus ook gelijk het manco van de aanpak. Als je ‘snelheid’-stappen neemt, dan kun je de wand al gepasseerd zijn voordat je je afvraagt of je een wand raakt. Nu treedt dat probleem op met een snelheid van rond de 30. Als ik de wanden wat dikker maak, zou het probleem dus pas bij hogere snelheden optreden. Op zich is dit natuurlijk mechanisch wel correct: op enig moment is een wand niet in staat een bewegend onderdeel vast te houden, afhankelijk van de energie die erin zit.
Onder (**2**) zie je de initialisatie. We maken het scherm schoon, we tekenen de wand, zie hieronder bij (**3**) en we gaan naar onze startpositie en stellen de richting en snelheid in. Ook hier zouden we iets met een random waarde kunnen doen, als mogelijke uitbreiding.
Onder (**3**) zie je de wand getekend worden. We hebben een andere kleur genomen voor horizontale en verticale wanden. Dit voorkomt andere, meer ingewikkelde, logica om te kijken hoe we de richting moeten veranderen. We kunnen het niet alleen van de richting afleiden, want je kunt met elke bewegingsrichting zowel tegen een wand als een vloer of een plafond stuiteren.
Bij (**4**) zie je het effect van een horizontale stuitering. We richten ons dan op 180 – ‘huidige richting’ en hebben deze gemodificeerd met een getal tussen -2 en +2. Deze formule zie je bij (**6**). De snelheid hebben we met 1 verhoogd, om de versnelling te laten optreden: daar was het om begonnen. Daarna hebben nog een blokje ‘herhaal tot’ ingevoegd, om te voorkomen dat bij het stuiteren het blokje binnen de wand blijft stuiteren. In het plaatje is die afvraag nog met de verkeerde kleur gebeurd; in mijn project heb ik dat aangepast.
Bij (**5**) zie je in feite dezelfde code als bij (4), alleen is de formule om de nieuwe richting te berekenen iets anders. Zoals hierboven vermeld: de nieuwe richting is de huidige richting keer -1.
Het programma is nog op veel manieren te verfraaien, maar in essentie is dit de versneller die me voor ogen stond. Alleen, het is mooier te maken, de box oogt wat ruw, er is geen geluid en de bal gaat een keer door de muur. Als we de muur te dik maken, blijft er weinig ruimte over om te stuiteren, maar kunnen we hogere snelheden halen. Zou er een andere manier zijn om tijdens het bewegen te achterhalen of de bal door de muur schiet? Het lijkt er ook op dat de bal door de wand schiet in de hoek waar de twee wanden samenkomen. Zou daar een oplossing voor kunnen zijn, zodat we dit gedrag kunnen voorkomen? Hoe zou het komen?



Opgave 15.2:

a. Hoe kun je voorkomen, dat de bal door de wand gaat?
b. Zit het probleem inderdaad in de hoeken? Zo ja, wat kun je daaraan doen? Zo nee, wat is dan het grote probleem?

Ten slotte nog dit. Ik wilde in het begin de achtergrond gewoon maken met lijntjes in plaats van een rechthoek. Achteraf realiseerde ik me dat je best wel met een rechthoek kunt werken en dan met het gummetje een gaatje kunt maken daar waar de bal eruit moet kunnen. Ofwel, het is niet nodig om in de code de box te maken, maar het maakt het natuurlijk wel mogelijk om elke keer de achtergrond een beetje te veranderen. In die zin is het dus wel een nuttige aanvulling.

< Figuur 7 - Totale code van de versneller