Ingenieursproject II: De Accelerometer

Verslag

Ruben De Facq - 01505950
Sean Deloddere - 01507227

Onder begeleiding van
Francis wyffels
21 mei 2017

Inhoud

1 De accelerometer

Raketten, defibrillators, airbags, een wii-afstandsbediening en zelfs de meeste smartphones. Al deze toestellen, hebben iets gemeenschappelijk, namelijk het een accelerometer. Een accelerometer meet versnellingen tegenover het object in rust. Wij hebben dit gebruikt om een spel te maken. Het leek ons het leukst om zelf een spel te bedenken en te ontwerpen in plaats van het vooropgestelde spel. Een ander spel maken bracht natuurlijk ook een andere probleemstelling met zich mee. Vandaar zal nu eerst even het concept van het spel toegelicht worden. Nadien zullen we de verschillende delen van het realiseren van het spel overlopen en toelichten hoe we ieder subprobleem hebben aangepakt. Tenslotte zullen we het nog hebben over enkele extra's die niet noodzakelijk zijn voor het spel, maar het wel aangenamer maken om te spelen.

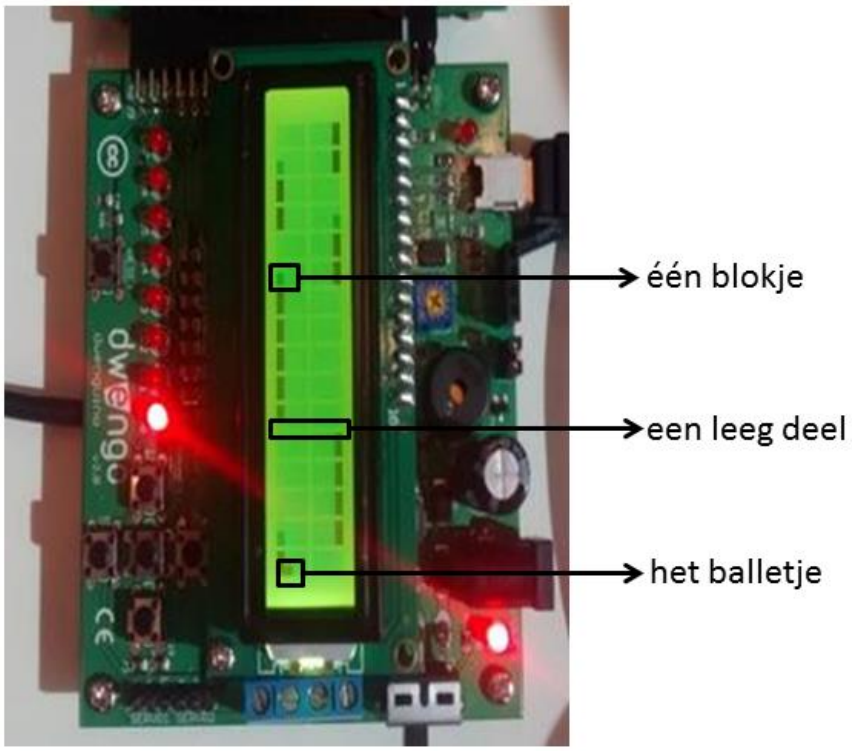
Voor het ontwerp van ons spel hebben we ons laten inspireren door games in de app store. Zo een spel moet niet ingewikkeld zijn. Belangrijk is dat het makkelijk te begrijpen is, vlot te spelen en een beetje verslavend. We wilden een spel maken waar de speler zelf, al spelend, de moeilijkheid kan bepalen, en dat willekeurigheid bevat zodat het spel oneindig veel opnieuw gespeeld kan worden maar toch nog steeds interessant aanvoelt voor de speler. Door een score in het spel te betrekken is er ook een licht verslavend element. Rekening houdend met al die elementen hebben we toen Ninja Jump ontworpen.

2 Ninja Jump

Ninja Jump is een spel waarbij de speler een ninja bestuurd die zich een weg baant door de lucht, door zich af te stoten op vallende objecten. We gebruiken een LCD met enkel zwarte pixels, daarom is de ninja vervangen door een bal en zijn de objecten vlakke blokjes.

Het scherm wordt verticaal gehouden tijdens het spelen, zoals weergeven in figuur 1. Aan de uiterst linker en rechterzijde vallen de blokjes naar beneden. De snelheid waarmee deze blokjes vallen kan door de speler zelf bepaald worden, door het toestel te draaien om zijn horizontale as. Onderaan het scherm bevindt zich het balletje, dat van links naar rechts en omgekeerd kan springen. Deze sprong wordt bestuurd door de speler, aan de hand van het roteren van het toestel om zijn verticale as, namelijk de kantelhoek.

De bedoeling van het spel is om zo een hoog mogelijke score te behalen binnen het tijdslimiet, zonder dood te gaan. Er wordt één punt toegekend per sprong. De rotatie om de horizontale as, namelijk de hellingshoek, is hier dus van groot belang. Door meer te roteren kan de speler de blokjes sneller doen vallen, waardoor hij meer heen en weer kan springen en meer punten kan halen. Een belangrijk design detail is dat niet enkel de blokjes sneller vallen, maar de bal ook sneller springt. Het versnellen en meer springen verhoogt de moeilijkheidsgraad en er is meer kans om een blokje te missen. Als dit gebeurt, of als de tijdslimiet verstreken is, verschijnt er een game over scherm. Hierop verschijnt de score, de plaats waar de speler is doodgegaan, en een tekst afhankelijk van of het spel is beëindigd door het missen van een blokje of het verlopen van de tijdslimiet.



Figuur 1: Het spel zoals te zien op de LCD. De bal is het kleine 2x2 vierkant onderaan. Er staan blokjes links en rechts die naar beneden vallen en waar de bal van de ene naar de andere kant moet springen. Er zijn ook lege delen waar de bal moet overspringen. Links zijn er acht lichtjes die de tijd weergeven.

3 Probleemstelling

De opdracht werd opgedeeld in delen om gestructureerd aan het werk te kunnen gaan. Vanzelfsprekend moesten eerst en vooral de waarden van de accelerometer worden ingelezen. Vervolgens moest het LCD scherm worden aangestuurd. Tenslotte stond de implementatie van het spel zelf op het programma. Ook dit kon weer in subproblemen opgesplitst worden.

De voornaamste delen in de spel lus zijn de volgende:
Als eerste hebben we een algoritme die een willekeurig level zal maken. Dan hebben we in de spel lus een manier om het spel sneller of trager te laten verlopen, het tekenen van het bord en het balletje op de juiste plaats, het balletje laten springen als de kantelhoek groot genoeg is en een test of het spel op zijn einde is. Er zijn ook nog extra implementaties, zoals het aansturen van lichtjes voor de hoeveelheid tijd dat nog over blijft, het registeren van knoppen om het spel te resetten… Elk deel zal in detail overlopen worden.

4 Verbinding accelerometer – dwenguinobord

Om de accelerometer te kunnen gebruiken, moet deze met het dwenguino bord verbonden worden. De verschillende connecties staan in figuur 2 weergeven. Het belangrijkste is om VCC met de 3.33V stroombron te verbinden. Ook moet de GND met de grond verbonden worden.

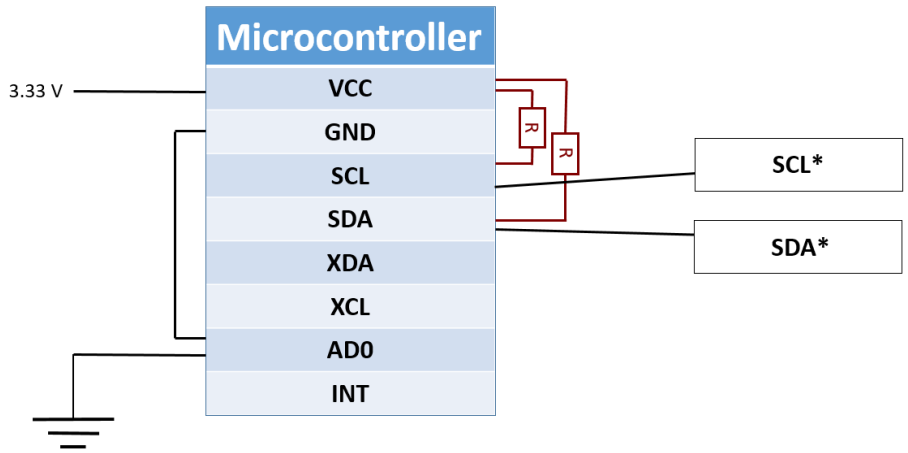
De andere verbindingen dienen voor het gebruik van de accelerometer. SCL en SDA dienen om met de accelerometer te communiceren. De SCL lijn is een klok, zodat het zenden van data correct getimed wordt. SDA is de data lijn. Merk dus op dat we maar één data lijn gebruiken en zoals nog zal worden uitgelegd, we een speciaal protocol met slave en master moeten gebruiken.

Om efficiënt te kunnen werken, moeten we de buslijnen hoog houden door middle van een pull-upweerstand. We gebruiken een weerstand van 2kΩ. AD0 wordt ook verbonden met de grond, zodat in het slaveadress de AD0 op 0 staat.

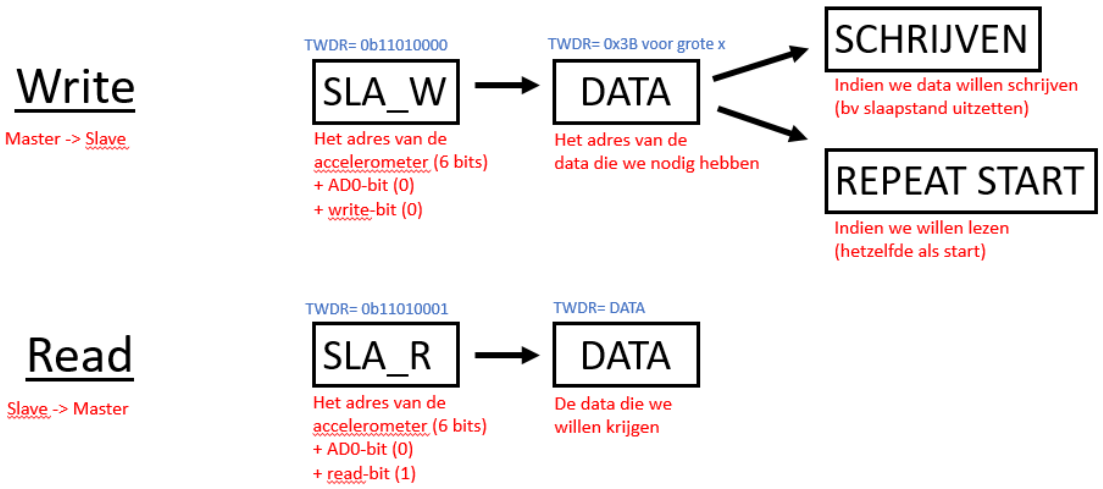
5 Versnellingswaarden lezen

Voor het inlezen van de waarde wordt gewerkt met een master-slave methode over één bus. In dit geval is het Dwenguino bord de master en de accelerometer de slave. We willen data uit onze slave naar de master sturen. Aan de hand van het procedure beschreven in de datasheet via I^2C protocol worden waarden ingelezen.

Voordat er ook maar iets met de accelerometer kan aangevangen worden, moet de slaapstand uitgezet worden. Vervolgens kan het protocol gestart worden. Eerst wordt een start conditie gestuurd en moet er telkens gewacht worden tot deze bit, of byte bij data, geaccepteerd is. Daarna wordt het slave adres doorgestuurd, zodat we met de juiste slave communiceren. Er kunnen namelijk verschillende slaves verbonden zijn met de master. Vervolgens wordt er geschreven naar de slave welke registerwaarde gelezen moet worden. Na een *repeat start* te sturen wordt dan een lees signaal naar de slave gestuurd, de master is dan klaar om de data te ontvangen. De data per x, y en z versnelling zijn 16-bit getallen en zijn gesplitst over twee registers van 8 bits. Om een volledige getal te lezen, wordt deze procedure twee keer herhaald.



Figuur 2: De bedrading van de microcontroller met het Dwenguinobord. Het midden toont de verschillende uitgangen van de microcontroller. Links staan de draden die met de stroombron te maken hebben, rechts hetgene dat zorgt voor de communicatie tussen het Dwenguinobord en de microcontroller.



Figuur 3: De algmene structuur om met de slave, namelijk de accelerometer, te communiceren. De eerste lijn geeft aan hoe je naar de slave schrijft en de tweede lijn hoe je ervan leest. *SLA* staat voor slave, met *W* voor schrijven en *R* voor lezen.

6 Het spel

De code van het spel bestaat uit vijf blokken: *Main*, *Tekenen*, *Print*, *Accelerometer* en *LCD*. In figuur 4 zijn alle blokken en hun relaties zichtbaar. Niet alle geimplementeerde functies zijn weergeven, enkel degene die gebruikt zullen worden om de structuur van de code uit te leggen.

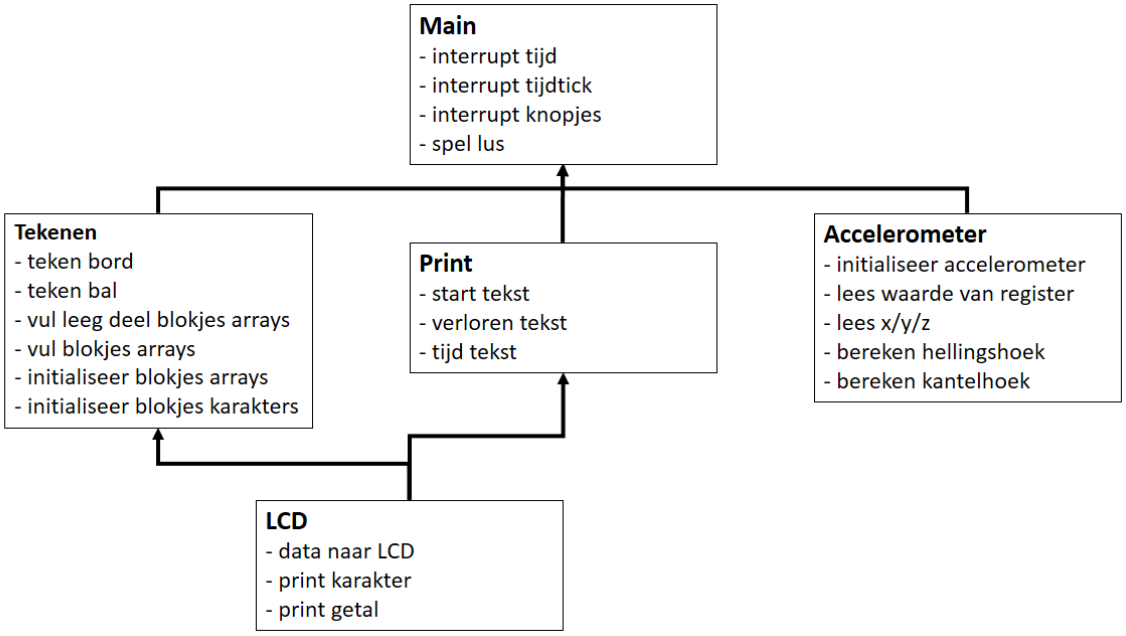
Tekenen staat in voor zowel het bord als de bal op de LCD te tekenen en om de gebruikte arrays die getekend worden, te initialiseren. Hiervoor gebruikt het dus de print functies van LCD. *Print* zal een bepaalde, voorgedefinieerde tekst printen op het scherm, met eventueel een extra variabele. Ook hier zijn de print functies van LCD nodig. Het blok *Accelerometer* staat in voor het lezen van de waarde van de accelerometer en deze om te zetten in een kantel- en hellingshoek. Het blok *Main* bevat verschillende interrupts om het spel precies te kunnen timen, om het induwen van knopjes te registeren en bevat de spel lus die alle aanpassing en nodige functies per tick zal oproepen.

Eerst zal elke blok in detail overlopen worden en eindigen we op het blok *Main*, waarin de spel lus wordt uitgelegd, namelijk hoe en in welke volgorde elke blok en hun functies gebruikt worden. De spel lus bevat ook een paar eigen functies.

6.1. LCD aansturen

De LCD wordt aangestuurd door gebruik te maken van de LCD library. Er wordt nog een extra functie toegevoegd om data naar de LCD te schrijven. Deze functie is bijna precies hetzelfde als een instructie sturen, maar de RS (Register select) pin wordt op low gezet, zodat de instructies geïnterpreteerd worden als karakter data.

Naast de basis ASCII-karakters, kunnen er acht extra karakters pixel per pixel geprogrammeerd worden. De LCD zal de karakters op het display aanpassen indien deze extra karakters aangepast worden. Dus ook op het display is er een beperking van acht verschillende extra karakters. Deze kunnen dan net zoals normale text, met de ASCII waarden van 0 tot en met 7, geprint worden met de al geimplementeerde functies.



Figuur 4: De algemene structuur van het spel. Er zijn vijf verschillende blokken. De pijlen tonen welke blokken van wie functies overerft om te gebruiken. Enkel de belangrijkste functies zijn opgesomd.

6.2. Accelerometer

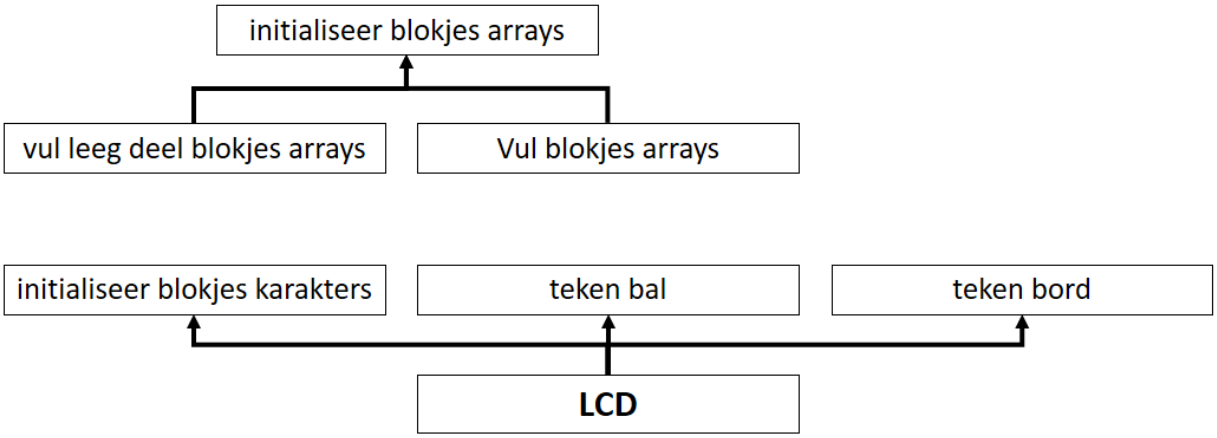
De blok *Accelerometer* bevat alle functies die nodig zijn om zowel de waarden van de accelerometer te lezen en deze om te zetten in bruikbare hoeken. De functie *initialiseer accelerometer* zal de registers van de accelerometer die instaan voor de slaapstand aanpassen, zodat de accelerometer op alle vlakken actief is. Dit is ook de enige keer dat we een waarde in het register van de accelerometer moeten aanpassen. Alle andere keren zullen we enkel waarden lezen.

De functie lees waarde van register zal de waarde van het opgegeven register lezen en deze teruggeven. Dit wordt gedaan met het protocol dat al vermeld werd. Lees x/y/z gebruikt deze functie om het correcte register te lezen. Zoals al vermeld, zal per coördinaat twee registers gelezen moeten worden, er bestaan dus in totaal zes van deze leesfuncties, namelijk één voor de grote en één voor de kleine waarden van de coördinaat.

Ten slotte bevat het ook de functies bereken hellingshoek en bereken kantelhoek. Deze functies gebruiken de leesfuncties om de nodige coordinaatwaarden te lezen en berekenen zo de hellingshoek of de kantelhoek in graden. Door gebruik te maken van simpele goniometrie kunnen de formules voor de kantelhoek en hellingshoek (in graden) bekomen worden:

$$\text{Kantelhoek} = \frac{180°}{\pi}\,\text{atan}\left(\frac{G_y}{\sqrt{{G_x}^2+{G_z}^2}}\right),$$

$$\text{Hellingshoek} = \frac{180°}{\pi}\,\text{atan}\left(\frac{-G_x}{G_z}\right).$$



Figuur 5: De strucuur van het blok *tekenen*. De drie bovenste functies hebben geen nood aan het blok *LCD* en staan in voor het level te initaliseren met een algoritme. *Teken bal* en *teken bord* zullen respectievelijk de bal en het bord op het LCD printen en *initialiseer blokjes karakters* zullen de nodige extra karakters initialiseren.

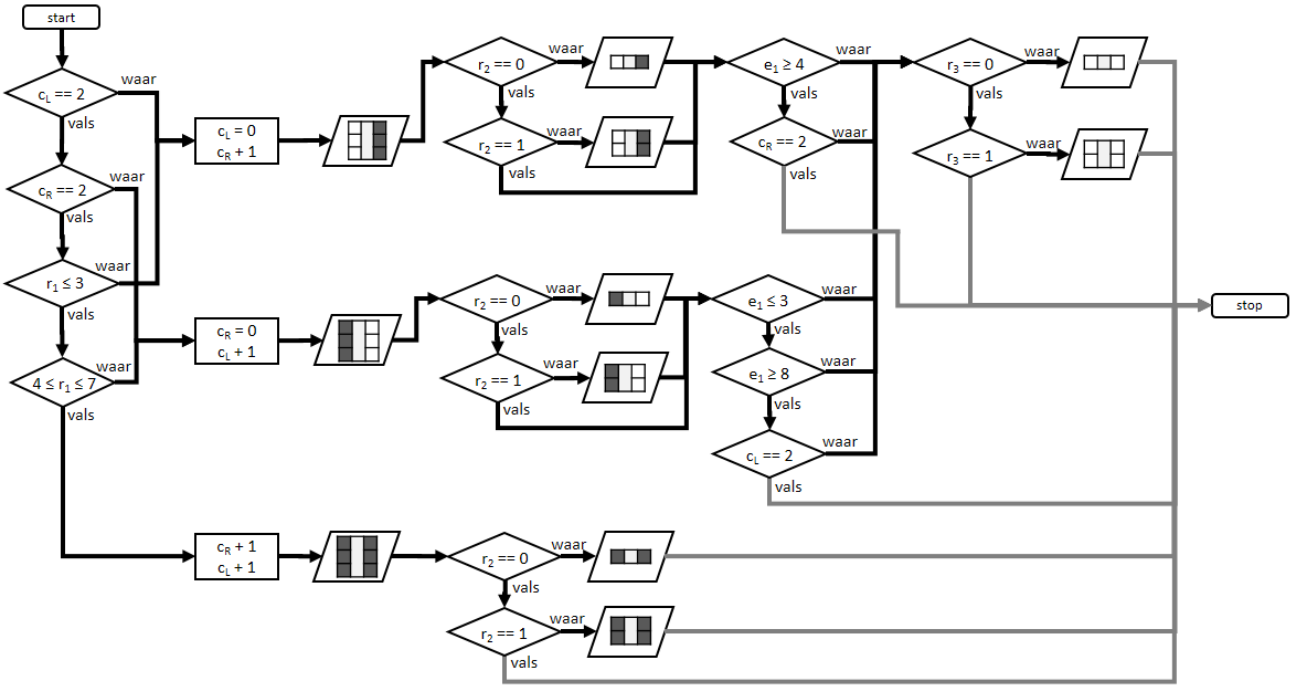
6.3. Tekenen

De structuur van het blok *tekenen* is weergeven in figuur 5. Deze bevat zowel de methodes om het level te initialiseren als om het bord en de blokjes op de LCD te printen.

Het spel is zo opgebouwd dat er voor zowel de blokjes links als voor de blokjes rechts een array wordt aangemaakt. Deze array is van een vooraf gedefinieerde lengte en bevat op elke index een 0 of 1. Een 0 staat voor een lege plaats en een 1 staat voor een blokje. Verschillende 1'en na elkaar zullen dus een langer blokje voorstellen, terwijl verschillende 0'en na elkaar een langere open plaats voorstellen, waar dus moet overgesprongen worden. Deze twee arrays bevatten het level van het spel. Elke vier ticks zullen de blokjes één plaats naar onder schuiven en in het bord gekopieerd worden. De arrays werken in een loop. Als de einde van de array is bereikt, zal terug het begin van de array gebruikt worden.

Deze twee arrays kunnen niet zomaar op een willekeurige manier ingevuld worden, omdat het level altijd speelbaar moet blijven. Als er een te lange open ruimte is, of als er aan de andere kant geen blokje is om naar toe te springen, kan het level onspeelbaar worden. Toch moet er een zeker willekeurigheid aan te pas komen, zodat het level altijd anders is. Tegelijk willen we ook te lange blokjes of te veel momenten waarop er blokjes aan beide kanten zijn vermijden, aangezien deze het spel triviaal makkelijk maken. Toch moet er een zeker willekeurigheid aan te pas komen, zodat het level altijd anders is.

Hiervoor wordt een algoritme gebruikt, die gebruik maakt van *leeg deel blokjes array* en *vul blokjes array*. *Leeg deel blokjes array* vult een nul, één of twee blokjes van beide arrays, zowel links als rechts, met 0'en. Dit stelt een open plaats voor. *Vul blokjes array* vult een bepaald interval van beide arrays met 1'en en 0'en, ofwel enkel links, ofwel enkel rechts, ofwel aan beide kanten. Welke dat er wordt gekozen, wordt meegegeven aan de functie. Waarom er een aparte functie is voor het lege deel en niet de *vul blokjes array* wordt gebruikt, wordt nog verduidelijkt in het algoritme.



Figuur 6: de algemene structuur van het algoritme voor het level. Diamond figuren staan voor een beslissing. Rechts is altijd het ja pad, naar onder het nee pad. In de rechthoeken staan aanpassingen aan variabelen en in de parallellogrammen de output, wat hierbij de blokjes arrays zijn. Deze zijn visueel voorgesteld. Één vierkant stel één getal voor, namelijk een 0 of een 1. Zwart zal een 1 invullen, terwijl wit een 0 invult. De licht grijze paden gaan altijd naar de stop, dus het einde van dit stroomdiagram. Aanpassingen aan eventuele externe variabelen, zoals het bijhouden van de index, worden niet weergeven. Het is verticaal opgedeeld in 8 subsecties. Subsectie 1 is bijvoorbeeld die 4 testen van c_L, c_R en r_1.

6.3.1. Het level algoritme

Figuur 6 toont het belangrijkste deel van het algoritme in een stroomschema waarvan *initialiseer blokjes arrays* gebruikt maakt. De initialisatie en het einde wordt niet getoont. Dit algoritme wordt

in een lus overlopen, tot de volledige array is opgevuld. Per iteratie door het algoritme, zal er een willekeurig aantal blokjes ingevuld worden. Na de stop zal dus terug aan de start beginnen, indien de array nog niet volledig gevuld is. De aanpassing van de index staat ook niet weergeven, maar moet dus tijdens het programmeren altijd incrementeren indien een blokje of lege plaats wordt toegevoegd. de parallellogrammen in subsectie 3 en 5 in figuur 6 maken gebruik van de functie *vul blokjes array*. Deze voegen blokjes toe op de plaats waar de array nog leeg is. Elk vierkant stelt één cijfer voor, namelijk een 0 of een 1, in de array. Bijvoorbeeld in subsectie 3, via de eerste parallellogram, zal aan de array voor de linker blokjes drie 0'en toegevoegd worden en aan de rechter array drie 1'en.

Het algoritme maakt gebruik van drie verschillende willekeurige waarden. De variabele r1 neemt een willekeurige waarde van 0 tot en met 8 aan. De variabele e1 is de waarde die r1 in de volgende iteratie zal aannemen. De waarden van deze willekeurige variabelen moeten dus altijd één stap voordat ze worden gebruikt, bepaald worden. Deze variabele bepaalt of we links, rechts of beiden blokjes zullen tekenen. De variabelen r_2 en r_3 nemen willekeurig de waarde 0, 1 of 2 aan, waarbij r_2 de lengte bepaalt van onze blokjes en r_3 de hoeveelheid lege plaatsen bepaalt die volgen op de

getekende blokjes, met een grootte van 0, 1 of 2. De variabelen c_L en c_R zijn tellers die starten op 0 bij het eerste maal uitvoeren van het algoritme. Ze houden bij hoeveel opeenvolgende keren er blokjes respectievelijk links en rechts getekend worden. We zullen per subsectie nog meer in detail bespreken wat iedere variabele doet.

Subsectie 7 en 8 stellen leeg deel blokjes array voor. Er kan enkel een leeg deel van lengte 0, 1 of 2 voorkomen in het spel, omdat het balletje niet over lengte van 3 kan springen. Het is dus interessant om dit in een aparte functie te definieren. Aan *Leeg deel blokjes array* wordt de willekeurige waarde r_3 meegegeven en zal het, afhankelijk van deze waarde, geen, één of twee lege blokjes toevoegen aan de array. Indien geen moeten toegevoegd worden, kan het oproepen van deze functie overgeslagen worden.

Het algoritme is zo opgebouwd, dat er altijd een speelbaar level wordt gegenereerd. Met r_1 wordt zoals eerder vermeld de keuze gemaakt tussen het vullen van enkel rechts, enkel links of beide kanten met blokjes. Er is $\frac{4}{9}$ voor enkel links, $\frac{4}{9}$ kans voor enkel rechts en $\frac{1}{9}$ kans voor aan beide kanten. Links en rechts moeten even vaak voorkomen. Blokjes aan beide kanten maakt het stukje van het level vrij gemakkelijk en dient dus om extra punten te halen met sprongen. Het is dan ook te verwachten dat de kans dat dit voorkomt, laag is. Naast deze kansen zal er ook voor de tak van enkel rechts gekozen worden, indien er in de vorige drie iteraties voor de tak enkel links is gekozen, en omgekeerd voor enkel links als er in de vorige drie iteraties enkel voor de tak enkel rechts is gekozen. Dit zorgt ervoor dat de kans op lange, niet interessante stukken met blokjes gevuld langs één kant onmogelijk zijn. Deze tellers overschrijven dus de willekeurigheid indien nodig. Het bijhouden en aanpassen van deze tellers staat in subsectie 2.

In subsectie 3 worden de arrays gevuld met blokjes. Subsectie 4 en 5 staan in voor het willekeurig maken van grootte van de opeenvolgende blokjes. De minimum lengte van een toegevoegd blokje is dus 3, waaraan dan nog 0, 1 of 2 blokjes worden bijgevoegd, met elk een kans van $\frac{1}{3}$. Subsectie 6 zorgt ervoor dat er enkel een leeg deel wordt aangevuld, indien dit mogelijk is voor het level. Er kan enkel over een leeg stuk gesprongen worden als er in het volgende deel een blokje aan de andere kant is. De variabele e_1 bevat de waarde van r_1 in de volgende iteratie. Het checkt dus of in de volgende iteratie aan de andere kant een blokje wordt geplaatst. De kant van de blokjes zou ook kunnen wisselen indien c_R of c_L de willekeurigheid overschrijven. Ook hiervoor moet dus gecheckt worden. Indien een van deze mogelijkheden zich voordoet, zal de methode *leeg deel blokjes array* opgeroepen worden, die 0, 1 of 2 lege blokjes bij de array invoegt, met elk $\frac{1}{3}$ kans.

Merk op dat de tak met blokjes aan beide zeiden niet kan eindigen met een leeg deel. Het zou mogelijk zijn om het spel te spelen, als dit wel het geval was, maar met testen was het duidelijk dat dit te moeilijk was om niet dood te gaan. De kans om een lengte van 3 blokjes is namelijk $\frac{1}{3}$. Als dit gebeurd, moet de speler op het onderste blokje terechtkomen en al direct weg springen bij het volgende. Vanaf de speler de snelheid omhoog laat gaan, is dit bijna onmogelijk om correct te doen en zorgt dit enkel voor frustraties. Er is dus bewust gekozen om de kans op dit voorval volledig te verwijderen.

Ten slotte moet de array ook een start en een einde hebben. De start is altijd een lengte van vier blokjes aan de linker kant. Het einde zijn altijd zes blokjes aan beide kanten. Dit is een stuk waar veel extra sprongen en dus extra punten kunnen behaald worden, wat nog eens een voordeel geeft om het spel op een hoge snelheid te spelen.

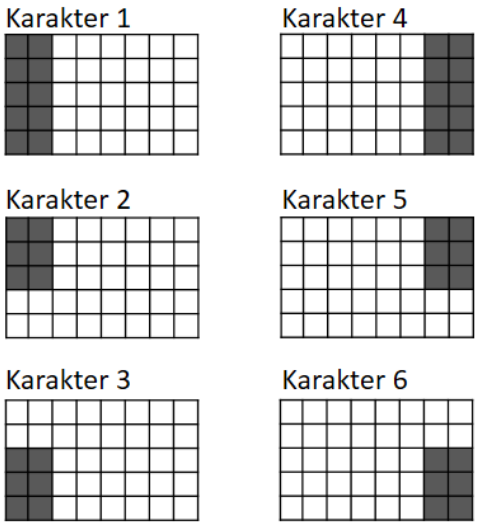
6.3.2. Karakters LCD

Initialiseer blokjes karakters zal zes van de acht extra karakters initialiseren, omdat deze altijd hetzelfde zijn. Er wordt voor elke kant drie karakters gemaakt: volledig blokje, bovenste deel blokje en onderste deel blokje. De bovenste vijftien karakters van het LCD zullen tijdens het spel altijd gevuld zijn met een van deze zes karakters, getekend met *teken bord* op het LCD. De twee onderste plaatsen bevatten ofwel ook een van deze karakters, of in combinatie met de bal, die via *teken bal* op het bord wordt getekend.

Een karakter van een LCD kan maximum 5 pixel breed en 8 pixels hoog. Omdat we ons scherm verticaal houden, wordt de breedte 8 pixels en de hoogte 5 pixels. In figuur 7 staan de zes karakters. Voor de blokjes wordt dus een dikte van 2 pixels genomen. Omdat we in de hoogte het karakter niet in twee kunnen verdelen, moet er een keuze gemaakt worden of we een blokje 2 of 3 pixels lang maken. Na de verschillende combinaties te testen, is er een duidelijk voordeel om te kiezen voor 3 pixels. Zo zijn ze altijd beter zichtbaar, wat belangrijk is als het spel op een hoge snelheid wordt gespeeld en het is zelfs duidelijker dat dit maar één blokje is.

6.3.3. Teken Functies

Er zijn twee aparte teken functies, één om het bord te tekenen en één voor het balletje. Zoals al uitgelegd, worden de waarden van de blokjes arrays in het bord gekopieerd, waarbij per 4 ticks de te kopiëren waarden één plaats naar onder schuiven. Om het spelbord te tekenen, wordt eerst teken bord opgeroepen, die het volledige LCD vult met de correcte balkjes. Daarna wordt teken bal opgeroepen, die enkel één van de onderste twee karakters van het LCD aanpast, zodat de bal ook wordt weergeven.



Figuur 7: Zes van de acht extra karakters die gebruikt worden, die de blokjes voorstellen. Deze moeten in het LCD geprogrammeerd worden, zodat ze via de ASCII waarden 0 tot en met 5 kunnen worden opgeroepen.

Teken bord zal alle logica behandelen die nodig is om voor de juiste karakter te kiezen om te printen. Dit is belangrijk, omdat we in de hele code eigenlijk werken met halve karakters, namelijk blokjes. *Teken bord* zet deze halve blokjes om in de juiste karakters, door om de twee waarden te kijken welke karakter moet getekend worden. Als bijvoorbeeld de eerste twee linkse waarden in bord van onder naar boven een 0 en een 1 zijn, zal teken bord karakter 2 printen op het onderste, linkse LCD-karakter.

Teken bal zal enkel één van de twee onderste LCD-karakters aanpassen. De bal bevindt zich altijd op één LCD-karakter. Het gebruikt dus maar één van de extra karakters. Omdat we acht extra karakters in totaal hebben, betekent dit dat we nog een extra karakter over hebben, die eventueel zou kunnen gebruikt worden voor een bepaalde uitbreiding, die wel zeer beperkt zou zijn. Deze functie gebruikt het bord om te weten welke soort balk er moet getekend worden en ook variabelen richting, bal en de tick. bal is een getal tussen 0 en 3 die laat wat de bal momenteel moet doen. Het getal 0 staat voor de bal die links stil staat en 3 als hij rechts stil staat. Als de bal in een sprong is, wordt 1 gebruikt indien de bal zich in de linkerhelft bevindt en 2 als hij zich in de rechterhelft bevindt. Richting wordt gebruikt om te weten in welke volgorde de sprong moet getekend worden. Als de bal zich op 1 bevindt, zal richting links aangeven als de bal van het midden naar de linker kant moet vliegen en rechts als de bal van het platform naar het midden moet vliegen. Uiteindelijk heeft de functie ook de tick nodig. De bal vliegt over één LCD-karakter in 4 ticks. De tick geeft dus aan hoe ver de bal in zijn sprong zit. In welke status de bal zich bevindt en welke richting het moet vliegen, staat in de spel lus.

6.4. Print

De blok *Print* bevat drie functies die een bepaalde tekst op het scherm printen. *Start tekst* wordt gebruikt aan de start van het spel. Deze print “DUW KNOP IN”. Het spel start namelijk pas als de knop naar boven wordt ingeduwd. *Tijd tekst* print “Tijd spel: “ met hierna een getal die toont hoe lang het spel zal duren in seconden. *Verloren tekst* print de game over tekst op het scherm met de behaalde score eronder. Als de speler is doodgaan, zal er “Verloren” geprint worden. Indien de speler tot aan het einde is geraakt, maar de tijd op was, zal de tekst “Tijd op” geprint worden.

6.5. Main

Uiteindelijk komen we tot de main, waar al deze functies samen komen om het spel te laten werken. De main bevat ook een paar eigen functies in de spel lus en interrupts. Eerst zullen de interrupts overlopen worden en het verschil tussen de tick en tijdtick. Daarna zal de spel lus deel per deel overlopen worden.

6.5.1. Interrupts

Er zijn drie interrupts voor de knoppen. Degene voor het noordelijk knopje wordt gebruikt aan het begin van het spel, om het level te laten starten en kan ook ingedrukt worden op het verloren scherm, om het spel de resetten, zodat opnieuw kan gespeeld worden met hetzelfde tijdslimiet. De twee andere interrupts zijn voor de oostelijk en westelijk knopje, die aan het begin van het spel kunnen gebruikt worden. Aan het begin van het spel kan je de tijd van het spel kiezen. Per druk op de westelijke knop, decrementeerd de tijd met één en per druk op oostelijke knop, incrementeerd de tijd met één.

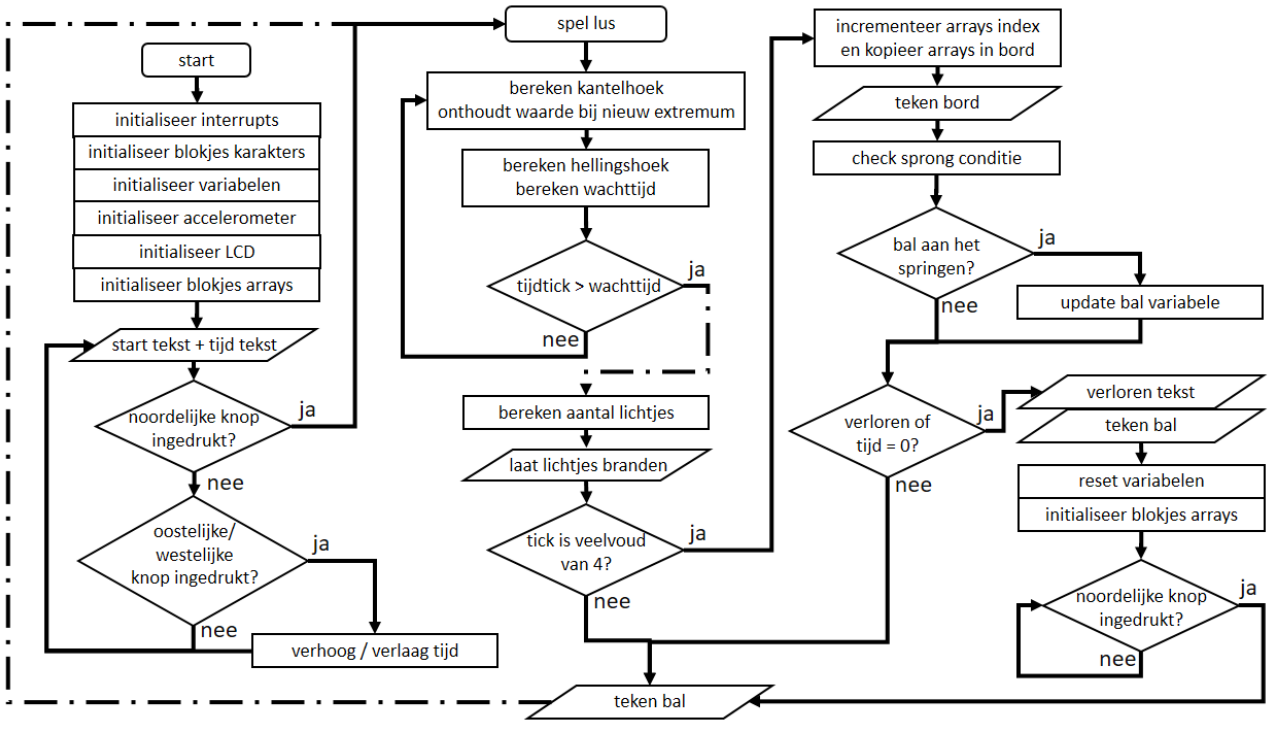
De interrupt voor tijd zal een variabele tijd per seconde met één verminderen. Deze wordt dus enkel gebruikt om het spel te laten stoppen als de tijd op is.

Uiteindelijk is er ook een interrupt voor de tijdtick. Deze zal een variabele tijdtick elke 8ms met één verhogen. Dit is de variabele die gebruikt wordt om alles in de spel lus te timen. Bepaalde berekeningen kunnen namelijk een verschillende lengte duren, zeker omdat niet altijd alles gebeurd. Zo duurt een tick van het spel altijd even lang, als op dezelfde snelheid wordt gespeeld.

Er is dus een belangrijk verschil tussen een tick en een tijdtick. Een tick heeft geen vaste tijd en zal dus van variabele lengte zijn. Per lus gaat er één tick voorbij. De tijdtick duurt precies 8ms en wordt gebruikt om het spel op een voorspelbare manier te laten verlopen. Elke deel van de lus duurt een een geheel aantal tijdticks, dus een veelvoud van 8ms. Zo hangt het spel niet af van hoeveel tijd een bepaalde berekening inneemt. Dit is belangrijk, omdat het spel kan versneld of vertraagd worden met de hellingshoek. Als er op een constante hellingshoek gebleven wordt, moet elke tick dezelfde tijdslengte hebben.

6.5.2. Spel lus

In de spel lus komen alle voorgaande functies, interrupts en logica samen met nog wat extra functies, om het spel functioneel te maken. Nu komt dus de echte kers op de taart.



Figuur 8: De structuur van de volledige main in een stroomdiagram. Merk op dat er nooit uit de spel lus wordt gegaan. Enkel de belangrijkste kenmerken zijn weergeven. Aanpassen van bepaalde variabelen of onderhoudt is niet weergeven. Een gestippeld pad stel een pad voor waarbij eerst wordt gewacht tot een tijdtick volledig is voorbijgegaan. Hierna wordt ook het aantal tijdticks op 0 gezet, zodat erna kan gemeten worden hoeveel tijdticks er voorbij gaan vanaf het einde van dit gestippeld pad. Dit gebeurd twee keer. Het eerste deel duurt een variabel aantal tijdticks, terwijl het tweede deel er altijd twee duurt.

in figuur 8 zijn de begin functies en de verschillende delen van de spel lus weergeven. Enkel de belangrijkste delen staan weergeven. Het aanpassen van bepaalde variabelen, zoals bijvoorbeeld tijdtick op 0 zetten voordat er een *wachttijd* mee berekend wordt, worden niet getoont. Bij de gestippelde pijlen wordt er gewacht tot een volledige tijdtick voorbij is geweest. Dit minimaal verlies van maximum 8ms waarbij de microcontroller niet doet, overtreft het voordeel dat het spel altijd over dezelfde verwachte snelheid werkt.

De main start met het initialiseren van verschillende componenten die we nodig hebben en zet alles klaar om een eerste spel te starten. *Initialiseer blokjes arrays* wordt dus opgeroepen. Daarna wordt een *start tekst en tijd tekst* weergeven. Er wordt in een lus gecheckt of de start knop of de knoppen voor het verhogen en verlagen van de tijd worden ingedrukt. Als de start knop wordt ingedrukt, zal de spel lus starten, waarna het hier nooit meer uit komt.

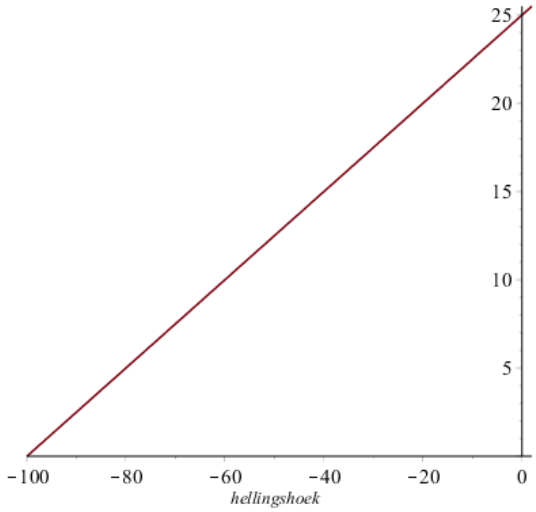
De spel lus is opgebouwd uit twee delen, één dat elke tick wordt uitgevoerd en één dat enkel elke 4 ticks wordt uitgevoerd. Merk op dat *teken bord* enkel elke 4 ticks wordt uitgevoerd en *teken bal* elke tick. Indien de bal zich in een sprong bevindt, duurt het 4 ticks om over één LCD-karakter te vliegen en dus 8 ticks om van de ene naar de andere kant te springen. De bal kan dus visueel elke tick veranderen als het zich in een sprong bevindt, maar het bord beweegt enkel om de 4 ticks.

Als eerste worden zowel de kantelhoek en hellingshoek berekend. Als de kantelhoek groot of klein genoeg wordt, wordt deze nieuwe waarde opgeslagen. De hoek dat dus gebruikt wordt in de rest van de lus, is de laatste hoek dat groot genoeg was om de bal te laten springen. De hellingshoek wordt gebruikt om met behulp van volgende formule de *wachttijd* te berekenen:

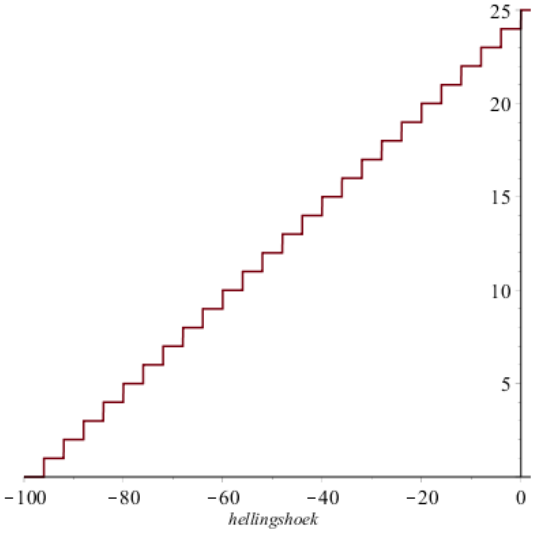
$$wachttijd = \frac{100+hellingshoek}{4}.$$

Merk op dat de hellingshoek altijd negatief is en zich tussen 0° en -99° bevindt. Een grotere negatieve helling zal in deze formule voor een kortere *wachttijd* zorgen. De *wachttijd* moet een natuurlijk getal zijn. Bij het delen door 4 gebruiken we een bitoperatie, wat zeer efficiënt is voor de microcontroller. Dit betekend ook dat de rest van de deling buiten beschouwing wordt gelaten. Figuur 9 toont deze formule met de precieze waarde, terwijl figuur 10 toont hoe het verloop in werkelijkheid is, namelijk als de rest wordt weggelaten. Dit is duidelijk een lineair verband, wat we nodig hebben. Bij grote waarden is het verschil in helling niet veel. Als we van 0° naar -20° gaan, wordt de snelheid maar ⅕ versneld, namelijk van *wachttijd* = 25 naar *wachttijd* = 20. Als we van -60° naar -80° gaan, wordt de snelheid verdubbeld, namelijk van *wachttijd* = 10 naar *wachttijd* = 5. Dit is het effect dat we verlangen.

Het blijft in deze lus tot dat de het aantal tijdticks groter wordt dan de wachttijd die berekend is met de hellingshoek. Er kan dus eerst een lage helling zijn, waardoor het lang zal wachten, maar tijdens het wachten kan de helling hoog gebracht worden om dus direct verder te gaan. Zo heeft de speler precieze controle over de snelheid van het spel, op elk moment. Als het aantal tijdticks de *wachttijd* overschrijdt, wordt nog gewacht tot de laatste tijdtick eindigd. Zo wordt er altijd een geheel aantal tijdticks gewacht en hangt de lengte van het wachten niet af van de berekeningen.



Figuur 9:
de wachttijd in functie van de hellingshoek, met rest.
De wachttijd is de verticale x-as.



Figuur 10:
de wachttijd in functie van de hellingshoek, zonder rest.
De wachttijd is de verticale x-as.

Dan worden het aantal lichtjes die moeten branden berekend. Dit kan op verschillende manieren, maar omdat de tijd verlaagd, werd gebruik gemaakt van een check:

$$tijd + tijdoffset < start\ tijd.$$

Als dit voldaan is, zal *tijdoffset* met $\frac{start\ tijd}{8}$ verhogen en een extra lampje beginnen branden. *Tijdoffset* begint op 0. Er kan gemakkelijk nagegaan worden dat een extra lampje zal branden elke keer dat er $\frac{start\ tijd}{8}$ seconden voorbij gaan. Na 8 keer zullen dus 8 lampjes branden en dit na exact start tijd seconden. Dit is maar één van de vele mogelijke implementaties om hetzelfde effect te krijgen.

Indien de tick geen veelvoud is van 4, wordt *teken bal* opgeroepen en start de spel lus opnieuw met de volgende tick. Indien dit wel het geval is, zal de index van de arrays verhogen, om dus de blokjes één plaats naar onder te laten bewegen en worden deze waarden in het bord gekopieerd. Daarna wordt de functie *teken bord* opgeroepen, dus het volledige LCD zal vullen met de correcte karakters.

Hierna wordt gecheckt of er een opgeslagen kantelhoek die groot genoeg is om de bal te laten springen, indien de bal op een blokje staat, en wordt de bal variabele geüpdatet. De variabele wordt ook geüpdatet als de bal al aan het springen was, zodat het bij de volgende tick zich op de volgende LCD-karakter zal bevinden. Indien een sprong zich voordoet, wordt de score met één verhoogd. Men moet de sprong dus niet afmaken om een punt te scoren, de sprong moet enkel starten.

Hierna wordt nagekeken of de bal op een blokje staat, indien het zicht niet in een sprong bevindt, of dat de tijd gelijk is aan 0. Indien dit niet het geval is, wordt teken bal opgeroepen en start de spel lus met de nieuwe tick. Als dit wel het geval is, wordt verloren tekst en teken bal opgeroepen. Zo kan de speler zien waarom het spel is gestopt, wat hun score is en waar de bal was op het einde en dus de reden voor het verliezen. Alle variabelen die nodig zijn in het spel, worden gereset. Er wordt een

nieuwe level aangemaakt door *initialiseer blokjes arrays* op te roepen. Daarna wordt er gewacht tot de knop wordt ingeduwd om een nieuw spel te starten. Het nieuwe spel duurt dezelfde tijd als de vorige, zodat scores met elkaar kunnen vergeleken worden.

7. Conclusie

Het design van het spel is door veel iteraties gegaan. Voor we op het gebruikte algoritme kwamen, hebben we nog drie andere mogelijke levels geprobeerd. Deze zagen er soms interessanter en meer willekeurig uit, maar om te spelen is deze laatste iteratie van het algoritme het leukst. Het zorgt ervoor dat de er veel gebruik kan gemaakt worden van de hellingshoek en is niet te gemakkelijk, noch te moeilijk.

Het spel zelf is vrij simpel, maar het is duidelijk dat er veel details aan te pas komen. Zo moet het level goed speelbaar zijn, maar ook willekeurig. Door het toevoegen van de tijdtick zal het spel altijd voorspelbaar zijn, wat zeker belangrijk is. Het kunnen versnellen en vertragen van het spel moet met de juiste hoeveelheid gebeuren en op een voorspelbare manier. Er moet een manier zijn om het level te resetten en nog vele andere kleine implementaties.

Dit verslag is gedetailleerd zodat zelf het spel kan gemaakt worden, maar is open genoeg om eigen design keuzes te kunnen kiezen. De figuren zijn zo opgebouwd dat deze zelfs genoeg zijn om het spel te programmeren. Dit is dus een naslagwerk dat kan gebruikt worden indien nodig.