



UNIVERSITEIT
GENT

Ingenieursproject II: Accelerometer

Antoine De Schrijver, Sebastiaan Verplancke

Begeleiders: Francis Wyffels, Tom Neutens, Alexander Vandesompele

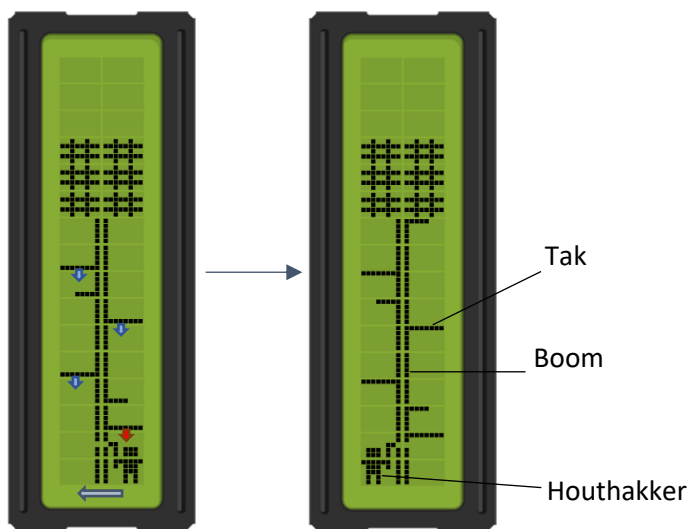
CONTENTS

1. Inleiding.....	3
2. Probleemstelling	4
2.1 Jack The Lumberjack	4
2.2 Besturing	4
2.3 Menu	4
3. Materiaal & Methode	5
3.1 Dwenguino	5
3.2 De LCD	5
3.3 De accelerometer.....	5
4. Resultaat	8
4.1 Structuur	8
4.2 Accelerometer.....	8
4.3 Lumberjack.....	10
4.4 Print.....	11
4.5 Input/Output.....	12
4.6 Main	12
5. Foutenanalyse	14
6. Conclusie	14
7. Dankwoord.....	14
8. Referenties.....	15

1. INLEIDING

Het doel van dit ingenieursproject is om aan de hand van een game te leren werken met 8-bit microcontrollers op C-niveau. Voor dit project werken we met een Dwenguino, een accelerometer en een paar weerstanden en kabels. De opdracht is om met de gegeven hardware een game te programmeren die gebruikt maakt van de accelerometer.

De game heeft de naam “Jack the Lumberjack” gekregen. Het concept van de game is simpel: een houthakker kan ofwel links, ofwel rechts van een boom staan. Hij kapt telkens het laagste stuk van de boom weg. De rest van de boom blijft overeind staan maar verlaagt wanneer er een stuk wordt gekapt. Het doel van het spel is om de houthakker zo te manoeuvreren dat de houthakker niet door de takken van de boom wordt geraakt. Het concept wordt visueel voorgesteld in figuur 1.



Figuur 1: Visuele voorstelling van Jack The Lumberjack. Elke keer dat de houthakker een stuk hakt verlaagt de rest van de boom. In de linker figuur zal de tak in de volgende iteratie op Jack vallen, dus verplaatst Jack de speler naar links. Dit is te zien op de rechterfiguur.

Alvorens het spel te kunnen programmeren moeten er een aantal voorbereidingen getroffen worden. Ten eerste moet men in staat zijn om de accelerometerdata uit te lezen. Vervolgens moeten er objecten op het scherm kunnen worden geprint. Daarna pas kan men overgaan naar het eigenlijke coderen.

In paragraaf 2 wordt het concept van het spel volledig uitgewerkt. De beschrijving van het gebruikte materiaal en de bijpassende methodes volgt in paragraaf 3. Vervolgens wordt dieper ingegaan op de code van het eigenlijke spel in paragraaf 4. Tot slot is er een reflectie aan de hand van de foutenanalyse in paragraaf 5 en de conclusie van dit project in paragraaf 6.

2. PROBLEEMSTELLING

2.1 Jack The Lumberjack

Zoals eerder vermeld is het doel van het spel om zolang mogelijk in leven te blijven en daardoor een zo hoog mogelijke score te behalen. Het spel wordt beëindigd als de *Health Points* van Jack op zijn. De health points worden voorgesteld door acht LED's. Jack begint het spel met alle 8 LEDs die branden. De *HP* kunnen op drie verschillende manieren naar beneden gaan. Ten eerste doordat de speler een lange tak niet weet te ontwijken en bijgevolg al zijn *HP* verliest. Ten tweede doordat de speler tegen een korte tak botst. In dit geval verliest hij 2 *HP*. De derde en laatste manier waarop Jack *HP* verliest is doordat er om de zoveel tijd één lampje uitgaat. Hier definiëren we ook meteen de moeilijkheidsgraad van het spel. Deze moeilijkheidsgraad gaat van 1 tot 5 en kan door de speler aangepast worden in het menu scherm. Hoe hoger de moeilijkheidsgraad, hoe korter de tijd tussen het verliezen van één *HP*. Men kan ook *HP* terugverdienen door een korte tak te ontwijken. Dan gaat er één LED terug aan.

Naast de *Health Points* krijgt Jack ook een score. Per lange tak die Jack weet te ontwijken wordt de score met 2 punten verhoogt. Per ontweken korte tak krijgt Jack 1 punt. Als een korte tak geraakt wordt verlaagd de score met 2 punten. De regels over *HP* en score worden samengevat in figuur 2.

2.2 Besturing







De speler kan Jack van kant bewegen door het Dwenguino bord over de horizontale as te kantelen. Om sneller te kappen, en zo ook sneller extra *HP* te krijgen, kan de speler het bord over de verticale as draaien. Hoe steiler het bord staat, hoe sneller de houthakker hakt en hoe sneller de takken naar beneden vallen. De bewegingen van het Dwenguino bord worden geregistreerd door de accelerometer.

2.3 Menu

Voordat de game gestart wordt krijgt de speler een menu te zien. Hier kan hij a.d.h.v. knoppen de moeilijkheidsgraad selecteren.

Als het spel *game over* is, krijgt de speler zijn score te zien en kan hij door het indrukken van een knop terugkeren naar het menu.

In de volgende paragraaf volgt een analyse van het materiaal en het toepassen van de methodes om dit spel te kunnen verwezenlijken.

	Korte Tak		Lange tak	
	Ontwijken	Raken	Ontwijken	Raken
HP 	+1  	-2  	/	-8 GAME OVER 
Score	+1	-2	+2	/

Figuur 2: Health point- en scoreberekening

3. MATERIAAL & METHODE

3.1 Dwenguino

Doorheen het project maken we gebruik van een Dwenguino. Dit is een microcontroller-bord van Dwengo.

3.1.1 De microcontroller

De microcontroller op de Dwenguino is een AT90USB646 van Atmel. Dit is een 8-bit CMOS-microcontroller gebaseerd op de *Atmel AVR enhanced RISC*-architectuur. De microcontroller heeft een maximale klok-frequentie van 8MHz aan 2,7V en 16MHz aan 4.5V. Een ander belangrijk kenmerk is dat de microcontroller het *two-wire interface* Master/Slave protocol ondersteunt. Dit komt uitgebreid aan bod in 3.3.4.

3.1.2 Het Dwenguino bord

Het bord zelf bevat onder andere 5 knoppen, een reset-knop, een LCD-aansluiting, een buzzer, 8 LEDs en analoge pinnen waar een extensiebord aan kan verbonden worden. Op dit extensiebord wordt onder andere de accelerometer toegevoegd.

3.2 De LCD

De LCD is een HD44780U van Hitachi. Het scherm is ingedeeld in 2x16 vakjes van 8x5 pixels. Het scherm wordt niet per pixel bestuurd, maar per karakter. Een karakter neemt de plaats in van 8x5 pixels. De LCD bezit ruim 200 vaste karakters. Voorbeelden zijn letters uit het alfabet en leestekens. Bovendien is er in het RAM-geheugen plaats voorzien voor 8 aangepaste karakters. Men kan dus 8 eigen karakters definiëren. De manier voor het aanmaken van eigen karakters wordt besproken in 0.

3.3 De accelerometer

De accelerometer is een MPU-6000 van InvenSense. Net zoals de microcontroller ondersteunt de accelerometer het *two-wire interface* Master/Slave protocol.

3.3.1 De basiswerking

Een accelerometer meet de versnelling, deze wordt meestal veroorzaakt door beweging. Als de accelerometer stilstaat is de enige data die de accelerometer voelt de valversnelling g van de aarde. De accelerometer uit dit project is een 3-axis accelerometer. Hij meet de versnelling over de x-as, de y-as en de z-as. Deze assen zijn rechtshandig

orthogonaal. Elke as zorgt dus voor eigen onafhankelijke data.

3.3.2 Registerwaarden

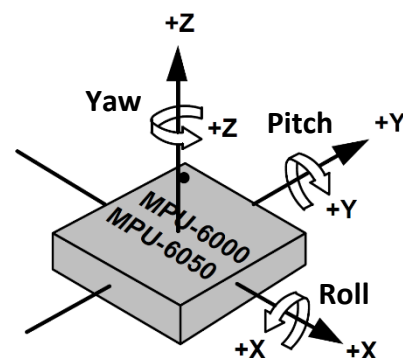
De waarde van elke as wordt als 16-bit signed integer opgeslagen. Dit 16-bit getal wordt opgesplitst in 2 registers: de 8 meest significante bits worden in één register opgeslagen, de 8 minst significante bits in een andere register. De exacte registers voor de verschillende assen worden weergegeven in Tabel 1.

Tabel 1. Registers voor accelerometerdata

	High	Low
x-as	0x3B	0x3C
y-as	0x3D	0x3E
z-as	0x3F	0x40

3.3.3 Interpretatie van de data

Uit de ruwe accelerometerdata valt er weinig concreet af te leiden. Deze data moet vertaald worden naar yaw, pitch en roll, zoals aangeduid op figuur 3. De accelerometerdata uit de registers moet dus in een functie gegoten worden om het aantal graden pitch/roll te bepalen.



Figuur 3: Visualisatie van assen met bijhorende yaw, pitch en roll

We veronderstellen voor de rest van de tekst dat als de accelerometer op een horizontaal oppervlak rust, de z-as naar boven wijst. Ten tweede veronderstellen we ook dat de accelerometer geen lineair acceleratie ondervindt.

Voor de volledige redenering achter het bekomen van de formules verwijzen we naar referentie [1]. Hieronder volgt alvast een korte samenvatting.

Het basisidee achter de formule is de volgende: men kan altijd één vector meten tegenover de assen: de gravitatievector. Deze vector duidt de valversnelling aan. Volgens de datasheet van de accelerometer heeft die gravitatievector enkel een z-component, met waarde 1, als de accelerometer op een horizontaal oppervlak rust.

Als de accelerometer niet in rust is heeft de gravitatievector een x-, y- en z-component volgens formule (1).

$$A = \begin{bmatrix} A_x \\ A_y \\ A_z \end{bmatrix} = R \cdot (g - a_{linear}) = R \cdot \left(\begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} - a_{linear} \right) \quad (1)$$

In deze formule is R de rotatiematrix die de gravitatievector beschrijft met het assenstel in het referentiestelsel van de aarde. Volgens onze veronderstelling is de lineaire acceleratie $a_{linear} = 0$, g is de gravitatievector gezien vanuit het perspectief van de aarde. A is de gravitatievector gemeten door de accelerometer.

De matrix R kan opgesplitst worden in 3 componenten; de roll, de pitch en de yaw, zoals aangeduid in figuur 3. De drie rotatiematrices worden gegeven door formule (2), (3) en (4).

$$R_x(\varphi) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\varphi) & \sin(\varphi) \\ 0 & -\sin(\varphi) & \cos(\varphi) \end{bmatrix} \quad (2)$$

$$R_y(\vartheta) = \begin{bmatrix} \cos(\vartheta) & 0 & -\sin(\vartheta) \\ 0 & 1 & 0 \\ \sin(\vartheta) & 0 & \cos(\vartheta) \end{bmatrix} \quad (3)$$

$$R_z(\psi) = \begin{bmatrix} \cos(\psi) & \sin(\psi) & 0 \\ -\sin(\psi) & \cos(\psi) & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (4)$$

Om R te bekomen moet men deze drie matrices met elkaar vermenigvuldigen. Aangezien matrixmultiplicatie geen commutatieve bewerking is, zijn er 6 verschillende multiplicaties mogelijk.

Als we de gravitatievector g vermenigvuldigen met R, moet de z-as altijd 1g zijn. We hebben dus 2 vrijheidsgraden. Dit zijn de roll en de pitch. De yaw kan niet tegelijkertijd met de roll en de pitch bepaald worden. Door deze voorwaarden blijven er twee matrixmultiplicaties over: formule (5) of formule (6).

$$R_{xyz} \cdot \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} = R_x \cdot R_y \cdot R_z \cdot \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} -\sin(\vartheta) \\ \cos(\vartheta)\sin(\varphi) \\ \cos(\vartheta)\cos(\varphi) \end{bmatrix} \quad (5)$$

$$R_{yxz} \cdot \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} = R_y \cdot R_x \cdot R_z \cdot \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} -\sin(\vartheta)\cos(\varphi) \\ \sin(\varphi) \\ \cos(\vartheta)\cos(\varphi) \end{bmatrix} \quad (6)$$

Voor de roll gaan we met formule (5) verder werken. We stellen de genormaliseerde accelerometerdata ($A/|A|$) gelijk aan het rechterlid van formule (5) en bekomen zo formule (7). Voor de pitch gaan we met formule (6) verder en bekomen we, analoog aan de roll, formule (8).

$$Pitch = \text{atan} \left(\frac{-A_x}{A_z} \right) \quad (8)$$

$$Roll = \text{atan} \left(\frac{A_y}{A_z} \right) \quad (7)$$

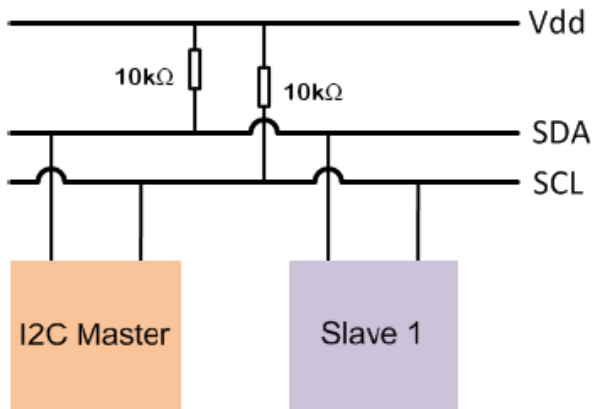
Bemerk dat deze formules de roll en de pitch in radialen geven. Als men naar graden wilt overgaan, moeten beide functies nog eens vermenigvuldigd worden met $180/\pi$.

3.3.4 Master/Slave Protocol

De communicatie tussen de Dwenguino en de accelerometer gebeurt aan de hand van het *two-wire interface* Master/Slave protocol. Dit protocol werd ontwikkeld door Philips onder de naam I²C. In dit protocol heerst een hiërarchie: één apparaat is de *master*, het andere is de *slave*. De *slave* is ondergeschikt aan de *master* en zal aan zijn instructies gehoorzamen.

Dit Master/Slave protocol maakt gebruik van een bus, op deze bus kan er één *master* zitten. Voor dit project is de Dwenguino de *master*. Aan diezelfde bus kunnen meerdere *slaves* toegevoegd worden. Hier is er slechts één *slave* nodig: de accelerometer.

De *two-wire interface* maakt gebruik van twee lijnen voor de communicatie tussen de *master* en de *slave*; de Serial Clock Line (SCL) en de Serial Data Line (SDA). De SCL of zorgt voor het kloksignaal. De SDA zorgt voor het datasignaal. Het verbindingsdiagram is zichtbaar op figuur 4. De Vdd-pin is de voedingsbron van ons netwerk. hierlangs wordt de nodige voltage geleverd.

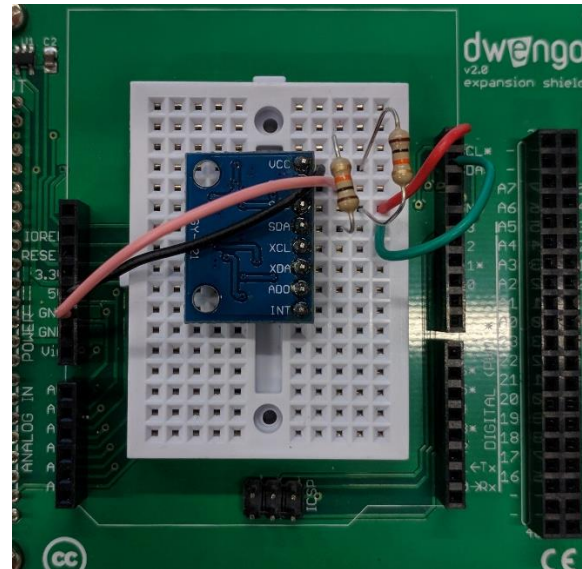


Figuur 4: Het two-wire interface verbindingsdiagramma

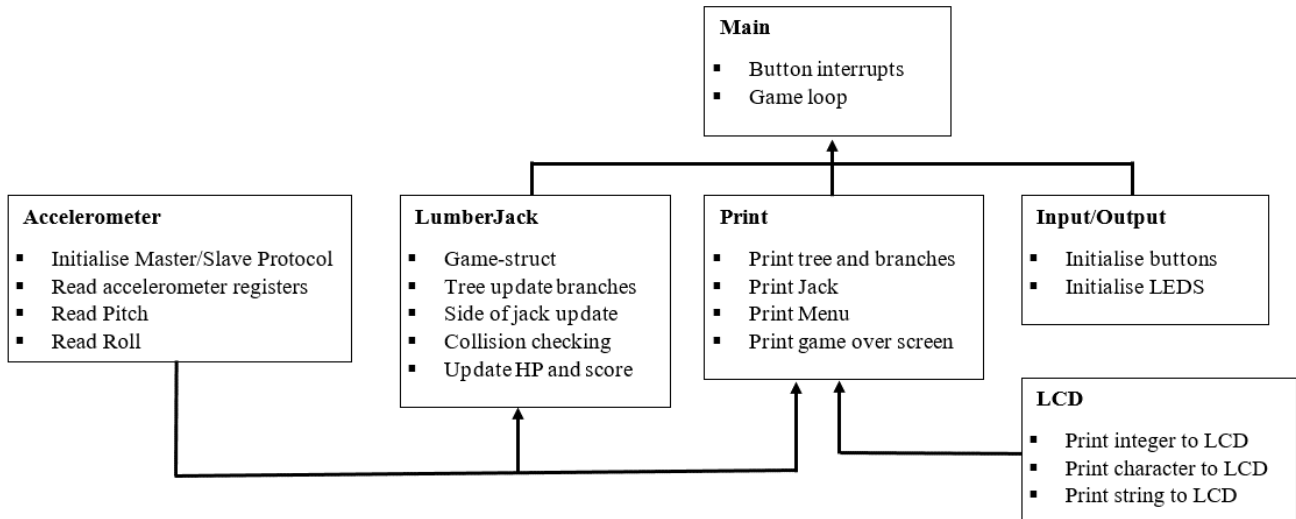
3.3.5 Hardware implementatie

De implementatie van het verbindingsdiagramma verloopt als volgt. Ten eerste wordt de ground (GND) van het bordje verbonden met de ground van de accelerometer. Analooog worden de Vdd's met elkaar verbonden. Dit gebeurt ook voor de SDA's en SCL's. Bij deze verbindingen worden ook twee externe pull-up weerstanden van 10kΩ gezet. Zo blijven de buslijnen hoog, zoals vermeld in de datasheet van de accelerometer. Als laatst wordt de AD0-pin van de accelerometer verbonden met de ground van de Dwenguino. Men had ook kunnen opteren voor het verbinden van de AD0 van de accelerometer met de Vdd van de Dwenguino. In dat geval is er ondersteuning voor het toevoegen van meerdere *slaves*. Voor dit project is dit echter niet nodig. Deze keuze wordt weerspiegeld in de code door de een bepaalde bit van het slave adres op 0 te plaatsen. Dit wordt verduidelijkt in figuur 7.

Een illustratie van de implementatie van het verbindingsdiagramma is te zien op figuur 5. De groene en rode draden verbinden respectievelijk de SDA en de SCL. De roze draad verbindt de ground met de ground. De zwarte draad verbindt de Vdd's met elkaar. Op het Dwenguino extensiebord wordt de Vdd aangeduid door "5V".



Figuur 5: Implementatie van het two-wire interface verbindingsdiagramma.



Figuur 6: Onderlinge samenhang van de verschillende componenten van het spel. Bij elke component worden de belangrijkste functies weergegeven. Een pijl naar een component significeert dat de wijzende component hier geïncorporeerd wordt.

4. RESULTAAT

Nu de verschillende methodes zijn uitgelegd, kan men eindelijk beginnen met het programmeren van de game zelf. Deze code is onderverdeeld in verschillende segmenten. Deze segmenten komen in deze paragraaf aan bod.

4.1 Structuur

Om het spel overzichtelijk te houden werd er gekozen voor een struct. Een struct is een datastructuur in C die toelaat om meerdere variabelen van een verschillend type bij te houden. Deze struct, die we Game hebben genoemd, houdt alle informatie van het spel bij. Vanuit de main wordt de struct als een pointer meegegeven aan de nodige functies. Zo kunnen deze functies gebruik maken van de game variabelen en deze desgewenst aanpassen. De struct wordt hieronder in pseudocode weergegeven. Voor de exacte betekenis van elke variabele verwijzen we naar de code-documentatie (referentie [6])

Pseudocode 1: de game-struct

```

struct Game {
    int** tree;
    int side_jack;
    int score;
    int counter;
    int difficulty;
    int state;
    int tilt;
    int speed;
};

```

Het spel is opgedeeld in 6 Secties: *Main*, *Accelerometer*, *Input/Output*, *Lumberjack*, *LCD* en *Print*. Het nut van elke sectie wordt in de volgende paragrafen besproken. Figuur 6 geeft de onderlinge samenhang tussen de blokken weer.

4.2 Accelerometer

Accelerometer bevat alle functies die nodig zijn om te communiceren met de accelerometer. Daarenboven bevat *Accelerometer* ook de functies die de ingelezen accelerometerdata omzetten naar de juiste hoeken a.d.h.v. de formules uit 3.3.3.

4.2.1 Master Slave Code

Voordat er data kan gelezen worden moet men de accelerometer eerst initialiseren a.d.h.v. een initialisatiefunctie. Eens de accelerometer geïntialiseerd is kan men op om het even welk moment data opvragen aan de hand van een readfunctie.

4.2.1.1 De Initialisatiefunctie

De pseudocode voor de initialisatiefunctie wordt hieronder weergegeven.

Pseudocode 2: De Initialisatiefunctie

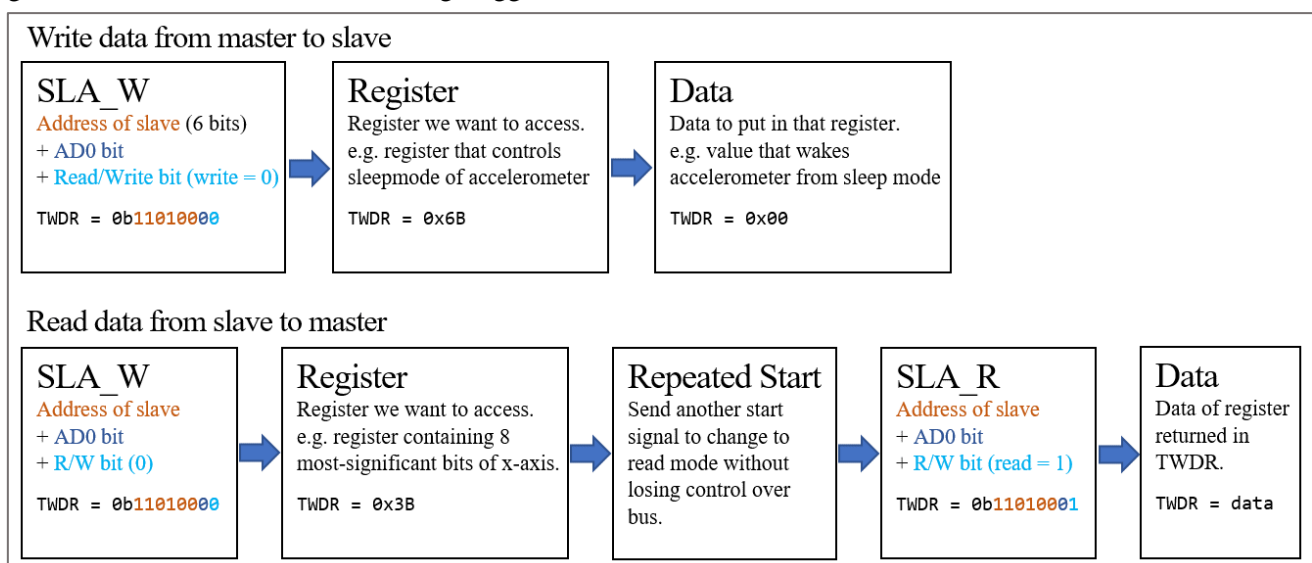
```
Initialisatiefunctie():  
    /*2-wire interface*/  
    2 PRR0 &= ~_BV(PRTWI)  
    /*startconditie*/  
    3 TCCR = _BV(TWINT) | _BV(TWSTA) |  
      _BV(TWEN);  
    /*wachten*/  
    4 while(!(TCCR & (1<<TWINT)));  
    /*errorcheck*/  
    5 if((TWSR & 0xF8) != statuscode) error();  
  
    6 0x00 naar register 0x6B schrijven om  
      accelerometer te doen ontwaken  
  
    /*stopconditie*/  
    7 TCCR = (1<<TWINT) | (1<<TWEN) | (1<<TWSTO);
```

Eerst wordt de *two-wire interface* aangeschakeld op regel 2 door de PRTWI-bit op 0 te zetten, zoals aangegeven in de microcontroller datasheet. Vervolgens moet men de data-overdracht initiëren. Hiervoor wordt een startsignaal doorgestuurd op regel 3 a.d.h.v. het *TWI Control Register* (TCCR). Vervolgens wacht men in regel 4 tot de verstuurde data is aangekomen door na te gaan of de *TWI interrupt flag* (TWINT) terug op 0 staat. De TWINT is de vlaggenbit: deze duidt aan of een proces bezig is of niet. Om na te gaan of de communicatie juist verlopen is wordt het *TWI Status Register* (TWSR) nagekeken op de juiste statuscode in regel 4. De specifieke statuscodes zijn in de microcontroller datasheet te vinden. Indien de statuscode niet de gewenste code is, wordt er een error getriggerd.

In regel 6 wil men schrijven naar de accelerometer. Het verloop van dit proces wordt geschetst in figuur 7. Eerst stopt men het adres van de slave gevolgd door een AD0- en een read/write-bit in het *Data and Address Shift Register* (TWDR). Het adres van de slave is het 6-bit adres van de accelerometer. Dit adres is te vinden in de datasheet van de accelerometer. De AD0-bit wordt op 0 gezet omdat er maar één slave is, zoals eerder besproken is. De read/write bit duidt aan of men wil schrijven of wil lezen. Het totaal van de 8 bits wordt SLA_W (slave_write) of SLA_R (slave_read) genoemd. Eens de SLA_W in de TWDR geladen is wordt de TCCR, analoog aan regel 3, aangepast om de transfer te beginnen. Analoog aan regel 4 en 5 wordt er gewacht en gecontroleerd of alles goed verlopen is.

Nu wordt het register naar waar men wil schrijven in de TWDR gestopt en doorgestuurd. Dit gebeurt op dezelfde manier als hierboven beschreven werd. Eens dit foutloos gebeurd is, wordt de data die men naar dat register wil schrijven, in de TWDR gestopt en doorgestuurd.

Om de initialisatieprocedure af te sluiten wordt in regel 7 een stopconditie doorgestuurd. De accelerometer is nu klaar voor gebruik.



Figuur 7: Read en Write verloop volgens het two-wire interface master/slave protocol

4.2.1.2 De Readfunctie

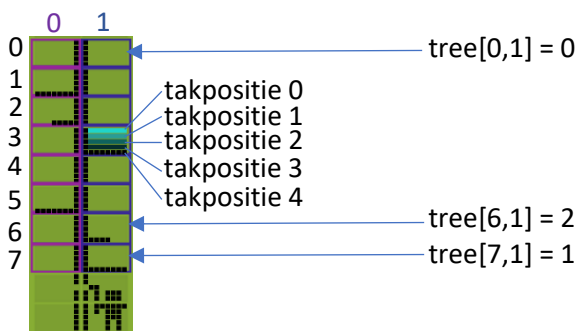
De readfunctie volgt grotendeels hetzelfde stramien als de initialisatiefunctie. Het grote verschil is dat men nu wil lezen i.p.v. schrijven. Het verloop van dit proces wordt geschetst in figuur 7. Eerst wordt een startconditie doorgestuurd. SLA_W wordt in TWDR gestopt en doorgestuurd. Vervolgens moet het register waaruit men de waarde wilt lezen in TWDR stoppen en doorsturen. Nu wordt er overgeschakeld naar de leesmodus. Dit gebeurt a.d.h.v. het sturen van een nieuwe startconditie. Deze is dezelfde als de eerste. Hierna wordt SLA_R in TWDR gestopt en doorgestuurd. Eens dit gebeurd is wordt de gewenste registerdata in TWDR teruggegeven. Bij dit project gaat het over acceleromterdata. Hoe deze data geïnterpreteerd moet worden, werd in 3.3.3 uitgelegd.

4.3 Lumberjack

Lumberjack bevat alle directe spelfuncties. Hier wordt er onder andere gecheckt op botsingen en scoreverhoging. Het updaten van de boom en het aanmaken van de *Game-struct* gebeuren hier ook.

4.3.1 Initialisatie

In dit blok gebeuren alle initialisaties van de variabelen in de game-struct. Hiervoor moet men eerst geheugen vrijmaken voor deze struct. Daarna worden de variabelen juist gezet en wordt er dynamisch geheugen vrijgemaakt voor de boom. Dit is de *tree* variabele. *Tree* is een matrix van 2 kolommen en 8 rijen die op iedere plaats een 0, 1 of 2 bevat. Een 0 staat voor een leeg vakje, een 1 voor een lange tak en een 2 voor een korte tak. Elke tak kan binnen een plaats 5 posities aannemen. De posities worden bijgehouden door de variabele *counter*. Dit wordt geïllustreerd in figuur 8.



Figuur 8: Weergave van de *tree* array met mogelijke waarden per arraypositie en mogelijke takposities binnen elke arraypositie.

4.3.2 Updates

4.3.2.1 Update Kant

Het eerste wat geüpdatet moet worden is aan welke kant Jack staat. Dit gebeurt in de functie *UpdateSide*. Deze functie stelt de variabele *side_jack* gelijk aan de macro LEFT als er genoeg naar links gekanteld is of aan RECHTS als er naar rechts gekanteld is. Als het aantal graden pitch niet verder is dan de PITCH_POINT dan blijft de variabele *side_jack* gelijk.

4.3.2.2 Update Tilt

In deze functie wordt de “steilheid” van het bord bijgewerkt. Afhankelijk van de roll hoek wordt de variabele *tilt* aangepast. Deze variabele wordt in de main gebruikt om te bepalen hoe snel de takken vallen.

4.3.2.3 Update Tree

Een tak kan op 5 plaatsen in een vakje zitten. Per conventie hebben alle takken in de boom dezelfde takpositie. Enkel als de tak op positie 4 is, zal hij de volgende val in een nieuw vakje terechtkomen. Als de tak in de bovenste 4 posities zit, zal er de volgende val niks speciaals gebeuren. De takpositie wordt dan gewoon met één verhoogt. Zoals eerder vermeld wordt de takpositie bijgehouden door de variabele *counter*. Aan de array *tree* wordt er dus niks veranderd als de counter 0-3 is. Als de counter gelijk is aan 4 verlagen alle takken een hokje en wordt er een nieuwe tak gegenereerd.

Om een nieuwe tak te genereren moeten er twee dingen gebeuren. Ten eerste moet er gekozen worden tussen een korte of een lange tak. Omdat Jack *HP* verkrijgt door korte takken te ontwijken, is ervoor gekozen om dubbel zoveel lange takken te genereren als korte. Een tweede factor is de plaats waarop de nieuwe tak moet komen. Dit kan links, rechts of helemaal nergens zijn. Hier heeft iedere optie evenveel kans om voor te komen. De randomfunctie die bepaalt of er een tak komt en welke soort tak het is, gebruikt een *seed* die afhangt van de pitch van het bord. Dit zorgt ervoor dat de boom bij elk nieuw spel verschillend is. 44,44% van de keren wordt dus een lange tak gegenereerd en 22,22% van de keren een korte tak. Er is dan nog 33,33% kans dat er niks wordt gegenereerd. Het is niet mogelijk om aan de linker- en de rechterkant in de array op dezelfde hoogte een tak te hebben.

Nadat er een nieuwe tak is gegenereerd wordt de boom geüpdatet. Dit gebeurt in omgekeerde volgorde van hoe de takken zakken. De laagste takposities (rij 7) worden eerst geüpdatet. We doen dit om te vermijden dat als een tak een blokje zakt, hij bij de update van het blokje eronder niet al aanwezig is en dus nog een keer zakt.

Het update-algoritme begint dus linksonder. Het kijkt voor iedere plaats in de array of er al dan niet een tak staat. Als er een tak is wordt de takpositie verhoogt. Als de takpositie 4 is, wordt de takpositie 0 en krijgt de arraypositie eronder de nieuwe tak (door op deze positie in *tree* de waarde 1 of 2 te plaatsen). In dit geval wordt de tak ook verwijderd van de oorspronkelijke arraypositie (door deze positie de waarde 0 te geven)

Als er helemaal vanonder, dus net boven Jack, nog een tak zit, wordt die gewoon verwijderd als Jack hem weet te ontwijken.

4.3.3 Checks

4.3.3.1 Check Collision

Om te weten of Jack de onderste tak al dan niet raakt, wordt de functie *CheckCollision* opgeroepen. Dit gebeurt net voordat de boom geüpdatet wordt. De counter staat in dat geval op 4, zodat de takken aan de onderkant van het vakje zitten.

Er zijn 2 parameters waarmee *CheckCollision* rekening moet houden: de kant van Jack en het soort tak onderaan in de array. Als Jack onder de tak staat is er een *collision*. Afhankelijk van welk soort tak worden de score en de *HP* aangepast. Bij een lange tak wordt de status van Jack ook op *DEAD* ingesteld.

4.3.3.2 Check Game Over

Dit is de laatste functie die besproken wordt in Lumberjack. Deze functie wordt net na *CheckCollision* opgeroepen. Hier wordt, als Jack dood is, alles klaargemaakt om naar het *GameOver*-scherm te gaan. Aangezien Jack niet enkel kan sterven door een lange tak te raken, maar ook doordat zijn *HP* op is, wordt dit ook nagegaan. Als het *Game Over* blijkt te zijn, wordt de *HP* op nul geplaatst en wordt *status* op *DEAD* gezet, moest dit nog niet gebeurd zijn in *CheckCollision*. Als laatste wordt de *tree* array leeg gemaakt. Dit zorgt ervoor dat de boom leeg is bij het starten van een nieuw spel.

4.4 Print

4.4.1 LCD-aansturing

Naast de accelerometer moeten we het scherm kunnen aansturen zoals we willen. We moeten dus in staat zijn om naast letters en cijfers ook eigen karakters op het scherm te tekenen.

We gebruiken de basis uit de gegeven *DwenguinoLCD* file om de karakters in het RAM-geheugen van de LCD te steken. Door het kopiëren van de standaardfunctie *commandLCD* en de *Register_Select* op *High* te zetten kunnen de eigen karakters doorgegeven worden en daarna geprint worden met de standaard printfunctie.

De moeilijkheid van de LCD zit in het feit dat er in totaal 32 plaatsen op het scherm zijn, maar dat er slechts 8 zelfgemaakte karakters in het RAM-geheugen passen. Elke keer dat het scherm vrijgemaakt wordt, kunnen er 8 nieuwe karakters gedefinieerd worden. Dit biedt ons de mogelijkheid om meer dan 8 karakters te hebben in het spel. Op elk moment kunnen er wel maar 8 op het scherm weergegeven worden. Dit is een enorme beperking op ons spel. De karakters zullen dus slim moeten gekozen worden.

4.4.2 Eigen karakters

Alles wat links van de boom geprint wordt, moet men ook rechts kunnen printen, maar dan in spiegelbeeld. Voor de rest van deze tekst zullen we verdergaan met enkel, de rechterkant.

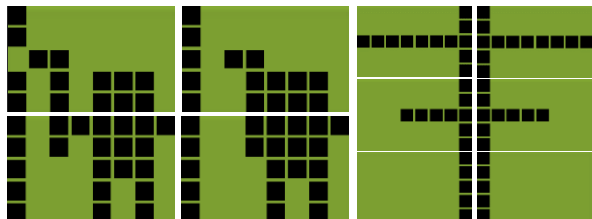
Het grootste probleem is dat we 5 verschillende lange takken, en 5 verschillende korte takken hebben. Elk vullen een verschillend lijntje in een karakter, afhankelijk van de takpositie. We zitten dus aan één kant al met 10 karakters. De oplossing van dit probleem is simpel: we zorgen ervoor dat er op elk moment slechts één mogelijke takpositie bestaat. Zo is er maar één karakter nodig voor een lange tak en één voor een korte tak. Elke keer dat er opnieuw naar het scherm geprint wordt, passen we dit karakter aan.

Naast de takken is er ook één karakter nodig voor een lege boom. Dit is een vast karakter die doorheen het spel niet aangepast moet worden.

Voor het mannetje, Jack, zijn er 2 karakters nodig: zijn bovenkant en zijn onderkant. Jack hoeft maar langs één kant te bestaan. Als het scherm ververs

wordt kunnen we net zoals bij de takken de twee Jack-karakters aanpassen. Naast het feit dat Jack links of rechts staat is er ook een animatie ingevoerd: jack zal de ene iteratie een gestrekte arm hebben en de andere iteratie een geplooid arm.

In totaal zijn er dus op elk moment 8 karakters: 2 jack-delen, 1 korte tak links en 1 korte tak rechts, 1 lange tak links en 1 lange tak rechts, 1 lege tak links en 1 lege tak rechts. De verschillende karakters worden weergegeven op figuur 9.



Figuur 9: De verschillende karakters van het spel. Jack heeft analoge karakters aan de linkerkant. De weergegeven takken hebben takpositie 2. Analoge karakters bestaan voor de andere posities.

4.4.3 Printfuncties

4.4.3.1 GenerateLumberJack

GenerateLumberJack zal de Jack-karakters aanpassen afhankelijk van of hij links of rechts staat. 3 tellen lang zal Jacks arm gestrekt zijn, en 2 tellen lang zal Jacks arm ingetrokken zijn. Dit geeft een mooie animatie bij het spelen. Merk op dat dit geen effect heeft op onze karakterlimiet: op elk moment bestaan er maar twee karakters voor Jack.

4.4.3.2 Generate(Small)Branch

GenerateBranch en *GenerateSmallBranch* zullen de takkarakters aanpassen afhankelijk van hun takpositie. Ze maken hiervoor gebruik van de variabele *counter*.

4.4.3.3 PrintGameToLCD

Deze functie zal de *tree* variabele gebruiken, de positie van Jack in rekening brengen en zo alle aangemaakte karakters op de juiste plaats printen op de LCD. Bovendien zal het boven de boom een kruin van hashtag-karakters printen. Dit karakter behoort tot de standaard-karakters en heeft dus geen effect op de karakterlimiet.

4.4.3.4 PrintMenu en PrintGameOver

PrintMenu zal het menu op het scherm afdrukken, het moet hiermee rekening houden met de *difficulty* variabele. Deze kan in het menuscherm aangepast

worden. Het doel van de *difficulty* werd in het begin van het verslag besproken.

PrintGameOver zal het *GameOver*-scherm op de LCD printen. Hierbij houdt het rekening met de variabele *score*. Het menu-scherm en het game-over scherm worden afgebeeld op figuur 10.



Figuur 10: Menu-scherm en gameover-scherm

4.5 Input/Output

De input/output sectie heeft maar 2 functies: *initLEDS* en *initButtons*. Deze functies maken de registers van respectievelijk de lichtjes en de knoppen klaar voor gebruik. Zo kan men in *Lumberjack* de LED's aanpassen en in de *Main* de button-interrupts invoeren.

4.6 Main

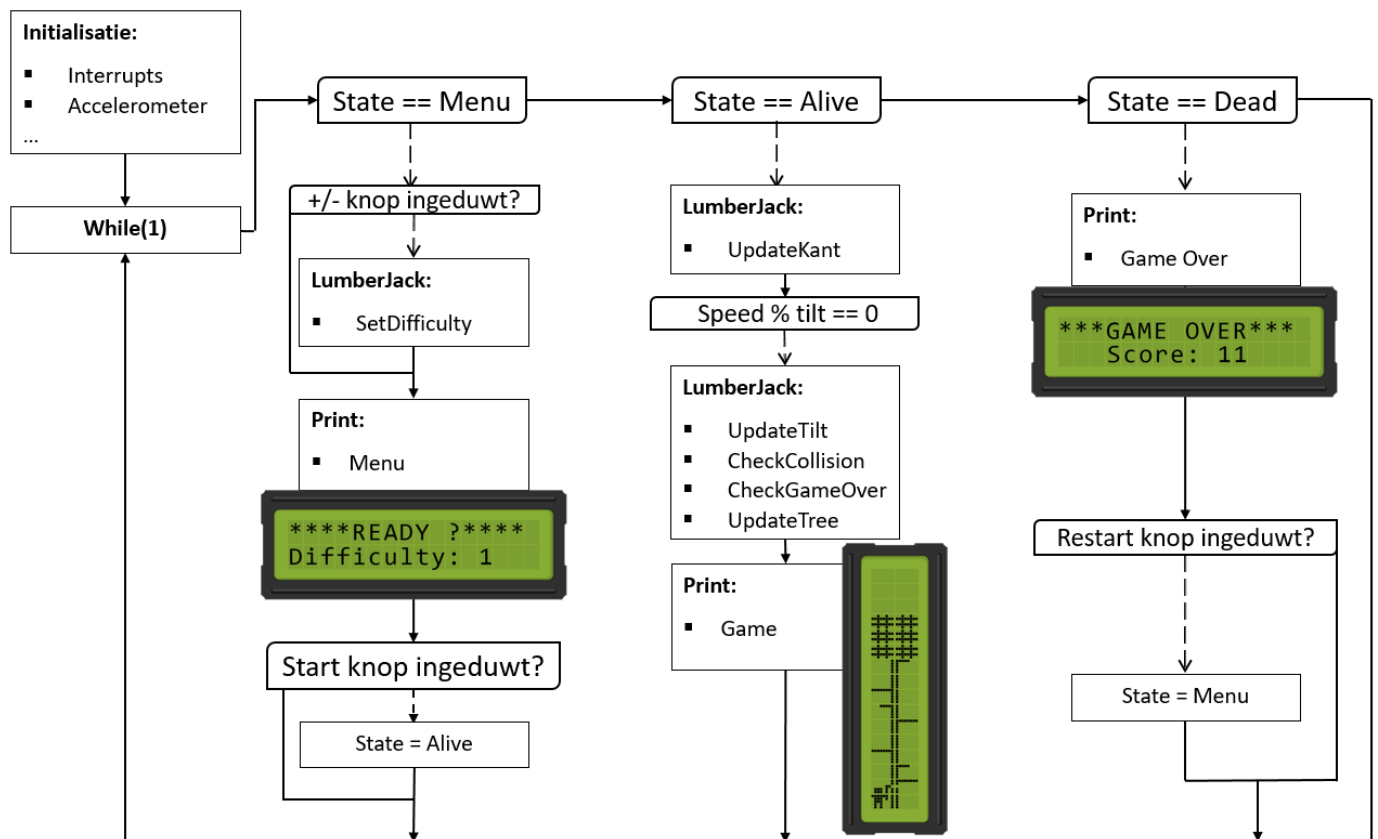
Nu alle andere secties klaar zijn voor gebruik, zal de *main* functie alles bij elkaar voegen en er zo voor zorgen dat correct verloopt.

4.6.1 Game-loop

De *main* functie maakt gebruik van een game-loop. Dit is een oneindige lus waarin alle nodige functies keer op keer worden opgeroepen.

Deze *main*-loop is iets complexer dan de klassieke *main*-loop: het bevat namelijk 3 loops die zullen blijven draaien, afhankelijk van de status van het spel: MENU, DEAD of ALIVE. De verschillende loops worden op figuur 11 weergegeven.

Om ervoor te zorgen dat niet alles elke loop geüpdatet wordt, maakt de loop gebruik van een teller genaamd *speed*. Deze teller wordt elke iteratie verhoogd. Als *speed* modulo een bepaald getal gelijk is aan 0, zal een bepaalde functie opgeroepen worden. Zo zal men om de 4 tellen de boom updaten, terwijl de kant van Jack om de tel wordt geüpdatet. Dit zorgt ervoor dat men Jack veel sneller kan bewegen dan de snelheid van de vallende takken. Het spel voelt hierdoor veel responsiever aan. De modulo-getallen hoeven niet vast te zijn: voor de takvalsnelheid wordt de variabele *tilt* gebruikt. Deze hangt af van hoe steil het bord staat, zodat een steiler bord resulteert in sneller vallende takken. Op dezelfde manier wordt om de modulo *difficulty* een LED uitgeschakeld.



Figuur 11: De Game-loop: de main begint bij de initialisatie, vervolgens start de while loop, hierin start de menu-loop. Deze blijft lopen tot als de speler de start knop indrukt. Eens dat gebeurd is komt het programma in de alive-loop terecht. Hier blijft het lopen tot het spel game over is. Op dat moment begint de state-dead loop. Deze blijft lopen tot de speler de reset knop indrukt. Dan zijn we weer aan het begin van het programma.

4.6.2 Interrupts

Naast de Game-loop zijn er ook bepaalde button-interrupts gedefinieerd. Deze interrupts laten toe om de *difficulty* aan te passen en de game te starten.

4.6.3 Begin en einde

Voor de game-loop worden verschillende initialisatiefuncties opgeroepen. Deze kwamen aan bod in de andere secties. Na de game-loop wordt ook de *Deletegame* functie opgeroepen. Deze functie zorgt voor het vrijmaken van het geheugen dat gealloceerd werd voor de game-struct. De game zal nooit buiten de loop komen, waardoor deze functie niet opgeroepen wordt. We hebben deze functie er voor de volledigheid toch bijgestoken.

5. FOUTENANALYSE

Doorheen het project hebben we met diverse problemen te maken gehad. Zo lukte het oorspronkelijk niet om accelerometerdata uit te lezen, terwijl we quasi zeker waren dat de code juist was. Dit bleek een hardware probleem te zijn met de voedingskabel van de Dwenguino. De initialisatiefunctie van *Accelerometer* wordt enkel correct uitgevoerd als de stroom via de USB-kabel komt, en niet via een stopcontact.

Een tweede fout was de aanname dat de ruwe accelerometerdata *unsigned* was terwijl deze in werkelijkheid *signed* was. Dit zorgde ervoor dat onze functies voor het bepalen van de pitch en roll graden oorspronkelijk niet werkten. Eens we de oorzaak doorhadden was de oplossing triviaal.

We hebben ook pas laat in het project vernomen dat er een karakterlimiet was. Initieel dachten we dat ons spel niet meer mogelijk was, maar zoals hierboven besproken hebben we, mits enige beperkingen en aanpassingen, ons doel bereikt. Eén zo'n aanpassing was de takdikte. Initieel wouden we de tak meer dan één pixel dik maken, om de valanimatie vloeiender te doen verlopen. Dit zou er echter voor gezorgd hebben dat we meer karakters nodig hadden dan mogelijk, aangezien een tak dan soms twee karakters nodig had. Een andere beperking was het weergeven van de score: oorspronkelijk wouden we de score bovenaan het spel meegeven. Omdat het bord gekanteld is bij het spelen, wouden we eigen karakters definiëren voor cijfers, zo konden deze in dezelfde oriëntatie als het spel gelezen worden. Vanwege de karakterlimiet was ook dit niet mogelijk. Als oplossing konden we de score verticaal weergeven, omdat daar cijferkarakters voor bestaan, maar dit was niet lelijk en onhandig. We hebben de live score dus weggelaten.

6. CONCLUSIE

Doorheen dit project hebben we verschillende vaardigheden aangeleerd. Dit is de eerste keer dat we met een microcontroller hebben gewerkt. De beperkte debug mogelijkheden en de kleinere reken capaciteit waren initieel een echte uitdaging.

Het begrijpen van een Master/Slave protocol zal ons in de toekomst ook kunnen helpen. Dit type protocol kent namelijk diverse toepassingen.

Voor het eerst in onze studies hebben we kennisgemaakt met datasheets. In het begin waren deze documenten zeer cryptisch, maar naarmate de tijd vorderde werden we beter in het begrijpen en selectief lezen van deze teksten.

Naast de technische aspecten van micro-controllers, datasheets en protocollen hebben we ook een eigen concept volledig leren uitwerken. Doorheen het project was er veel ruimte voor eigen invulling. Dit maakte het resultaat des te unieker en de voldoening op het einde des te groter.

7. DANKWOORD

Dit project hebben we niet alleen gemaakt. Doorheen het project konden we namelijk altijd rekenen op onze begeleiders: Francis Wyffels, Tom Neutens en Alexander Vandesompele. Zij hebben ons aangezet tot creatief probleemoplossend denken en hielpen ons verder bij grote obstakels. Zonder hen waren we hoogstwaarschijnlijk niet zeer ver geraakt in dit project.

8. REFERENTIES

- [1] Mark Pedley, Tilt Sensing Using a Three-Axis Accelerometer, <https://www.nxp.com>, 2013
- [2] Atmel Corporation, AT90USB646 8-bit Atmel Microcontroller with 64/128Kbytes of ISP Flash and USB Controller, 2012
- [3] Ronak Chokshi, RS-MPU-6000A Datasheet, 2013
- [4] Hitachi, HD44780U (LCD-II), 1999
- [5] InvenSense Inc., MPU-6000 and MPU-6050 product specification revision 3.4
- [6] Antoine De Schrijver, Jack The Lumberjack Code Annotation, 2018

Doorheen het verslag werden verschillende figuren gebruikt die aanpassingen zijn van bestaande figuren. De oorspronkelijke figuren worden hier vermeld:

- [1] Figuur 3: gebaseerd op figuur uit referentie [3]
- [2] Figuur 4: gebaseerd op figuur 2 uit de opgave
- [3] Figuur 10: gebaseerd op figuur van <http://haneefputtur.com>