**Leander Clappaert**

**2 ICT 4**

**Wetenschappelijke toepassingen 1 – Opdracht 1 – Geluid**

Inhoudstafel

[Inleiding 1](#_Toc438857535)

[Inlezen en coderen 2](#_Toc438857536)

[Setup en aanmaken \*.wav bestand 2](#_Toc438857537)

[Het \*.wav bestand theoretisch bekeken 3](#_Toc438857538)

[Aanmaken Data entities en gebruik datatypes 3](#_Toc438857539)

[Data entities 3](#_Toc438857540)

[Datatypes 4](#_Toc438857541)

[Genereren van wave arrays en invullen ontbrekende informatie 4](#_Toc438857542)

[Samples en waves 4](#_Toc438857543)

[Ontbrekende informatie 4](#_Toc438857544)

[Wegschrijven naar een \*.wav bestand 5](#_Toc438857545)

[GUI en features 5](#_Toc438857546)

[Failed experiments 5](#_Toc438857547)

[Problemen aanmaken \*.wav bestand 5](#_Toc438857548)

[Overige problemen 6](#_Toc438857549)

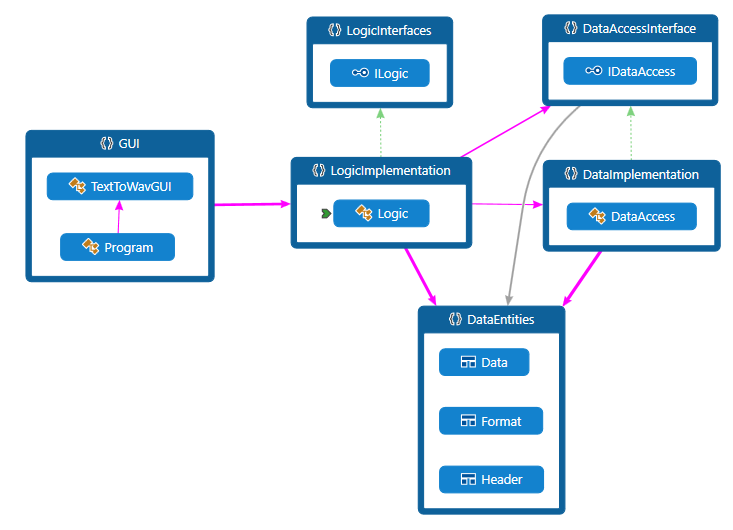
[Besluit 7](#_Toc438857550)

[Bibliografie 7](#_Toc438857551)

# Inleiding

De toepassing die dient gemaakt te worden is een programma dat tekst omzet naar geluid. Het geluid bestaat uit sinusoïden die opgeslagen worden in een \*.wav bestand.

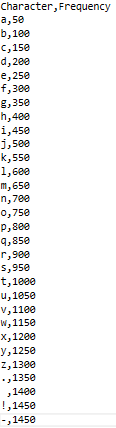
De opdracht bestaat uit 3 grote delen: het inlezen van tekst en coderen, het berekenen van elk onderdeel van het \*.wav bestand en het effectief wegschrijven van die onderdelen in één bestand.



IMG1: overview program

# Inlezen en coderen

De tekst wordt ingelezen als string en wordt vervolgens gecheckt. Indien er foute characters aanwezig zijn, die bijgevolg niet kunnen gecodeerd worden, zal de huidige tekst aangepast moeten worden door de gebruiker. Elke letter van de ingevoerde tekst wordt vergeleken met de CsvCharacters array (zie later). Indien 1 van de letters er niet in voorkomt, zal de gebruiker een nieuwe tekst moeten invoeren.

Indien de tekst goed is, zal deze gecodeerd worden aan de hand van een \*.csv bestand.

Het \*.csv bestand (= Comma-Separated Values) is een specificatie voor tabelbestanden. De waarden worden gescheiden door komma’s en je kan er gemakkelijk verscheidene tabellen uithalen. In deze opdracht worden er in het \*.csv bestand 2 tabellen bijgehouden: Character en Frequency.

Doordat de code opgeslagen wordt in een \*.csv bestand, zullen alle veranderingen aan de frequenties en aan de characters onmiddelllijk geïmplementeerd worden. Dit heeft invloed op de controle bij tekstinvoer, alsook de frequenties die doorgestuurd worden naar de logica. Bij de DataAccess zijn er 2 methodes die deze tabellen uitlezen en opslaan in een array: CsvCharacters en CsvFrequencies.

De ingevoerde en reeds gecontroleerde tekst wordt in de methode InputFrequencies vergeleken met de CsvCharacters array. Elke letter van de tekst komt overeen op een bepaalde positie binnen de CsvCharacters array. Die positie wordt vergeleken met de CsvFrequencies array om zo de frequentiewaarde van die letter te weten te komen. De bekomen frequentiewaarde wordt opgeslagen in een nieuwe array. De array is dus als het ware een gecodeerde versie van de huidige ingevoerde tekst.

Deze gecodeerde array zal gebruikt worden om voor elke letter een sinus te genereren (zie volgende sectie).

IMG2: Snapshot \*.csv file

# Setup en aanmaken \*.wav bestand

Een \*.wav bestand bestaat uit 3 grote onderdelen: een header-, een format- en een datagedeelte. Het formaat moet exact overeenkomen met de vooropgestelde definite, anders zal een audio/media programma nooit het bestand kunnen lezen en dus bijgevolg nooit afgespeeld kunnen worden. Om een goed bestand te genereren, moest er eerst goed begrepen worden hoe een \*.wav bestand werkt.

## Het \*.wav bestand theoretisch bekeken

Het is niet de bedoeling om alles zeer gedetailleerd uiteen te zetten, maar er worden enkele belangrijke punten aangehaald waar rekening mee moest gehouden worden.

De header bestaat uit 3 *chunks*, de format uit 8 en de data tevens uit 3 chunks.

Er zijn enkele vaste chunks die op voorhand bepaald zijn om zo een goede werking van het programma te garanderen: ChunkID, Format, Subchunk1ID, Subchunk1Size, AudioFormat, en Subchunk2ID.

Enkele chunks krijgen een vaste waarde toegewezen die niet beïnvloedbaar zijn door andere onderdelen of andere values nodig hebben: Subchunk1Size, AudioFormat, NumChannels, SampleRate en BitsPerSample.

Alle andere chunks - ChunkSize, ByteRate, BlockAlign, Subchunk2size en Data - zijn bijgevolg beïnvloedbaar en worden vervolgens tevens berekend.

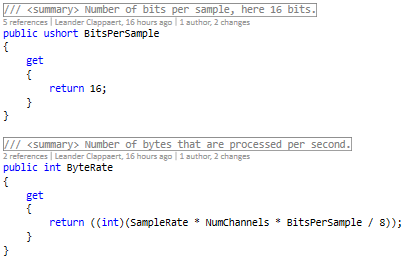
IMG3: \*.wav file

## Aanmaken Data entities en gebruik datatypes

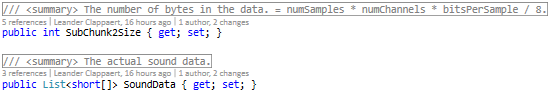
### Data entities

Om alle gegevens goed te structureren, om een gemakkelijkere en vooral efficiënte flow van het programma te garanderen, is er van elk van de 3 grote onderdelen van het \*.wav bestand een entiteit aangemaakt. Er is een struct Header, een struct Format en een struct Data. Er wordt een struct gebruikt in de plaats van een klasse

Elk onderdeel bevat de nodige methodes met informatie om de waarde van elke chunk te getten[[1]](#footnote-1), te setten[[2]](#footnote-2) of soms beiden te doen. IMG3 en IMG4 tonen 3 soorten methodes: een getter waarbij ik zelf reeds een waarde instel, een getter waarbij de returnwaarde berekend wordt aan de hand van een formule en een methode die eerst geset moet worden alvorens deze geget kan worden.



IMG4: getter methodes



IMG5: get/set methode

### Datatypes

De datatypes die gebruikt worden zijn allemaal gebaseerd op research naar goede voorbeelden op het internet. Vaak wordt er gebruikt gemaakt van een uint, ushort... Dit is een unsigned type. Dit betekent dat de laagste instelbare waarde 0 is en de hoogst mogelijke waarde het dubbele is van een normaal type (int, short). Ushort wordt gebruikt wanneer de waarde kleiner is dan 65535. Wanneer dit niet te garanderen is, wordt er een groter unsigned type gebruikt (uint, ulong...).

## Genereren van wave arrays en invullen ontbrekende informatie

### Samples en waves

Het bepalen van het aantal samples en het aanmaken van random waves is nagenoeg volledig gebaseerd op een blog artikel van microsoft[[3]](#footnote-3).

Het aantal samples dat genomen wordt per wave, wordt bepaald door de sample rate en het aantal channels[[4]](#footnote-4). Dit sample aantal zal bepalen hoe groot elke array is, dit voor iedere character. Omdat een gigantische array waarin alle letters op rij bewaard worden omslachtig is, wordt elke array afzonderlijk in een list opgeslagen. Hierdoor zal elke array afzonderlijk worden afgeprint en resulteert dit in hetzelfde resultaat. Echter nemen deze berekeningen minder rekenkracht in omdat we met kleine, afzonderlijke arrays werken die, wanneer ze volledig gevuld zijn, gewoon worden weggeschreven naar een list.

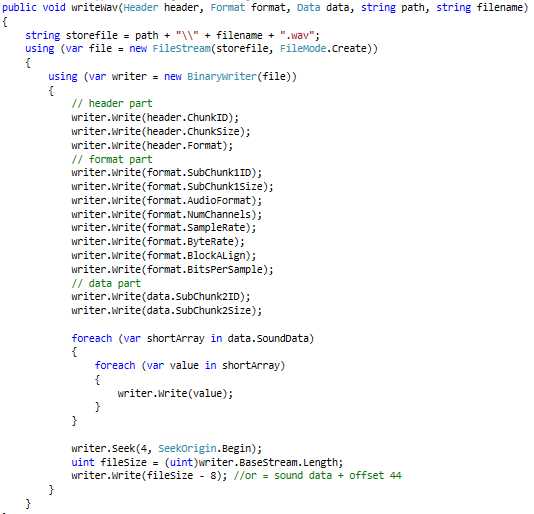
De samples worden aan de hand van een driedubbele for-loop gecreeërd. De eerste for-loop overloopt elke frequentie uit de gecodeerde frequentie array. Elke van deze frequenties wordt gebruikt om de hoek ϑ van de sinusfunctie te bepalen[[5]](#footnote-5). De tweede for-loop zal elke positie van de sample array overlopen. De laatste for-loop zal deze array effectief vullen met een sinusfunctie voor elke positie van de array. Vervolgens keren we terug naar de eerste for-loop en storen we de array opnieuw in de list. Dit gaat zo door totdat alle frequenties overlopen zijn.

### Ontbrekende informatie

Nu dat alle data berekend is, kan alle ontbrekende informatie aangevuld worden in de Data Entities. De Subchunk2Size, de ChunkSize en de data zelf. Deze worden opgeslagen aan de hand van de mogelijkheid om die methodes te setten. Zoals reeds vermeld, alle andere methodes bevatten geen set mogelijkheid, waardoor deze bijgevolg niet beïnvloedbaar zijn.

## Wegschrijven naar een \*.wav bestand

Het wegschrijven van alle nodige data naar het bestand gebeurt in de DataAccess.

Er wordt eenmalig een stream aangemaakt met de path waarnaar geschreven wordt. Elke lijn wordt binair naar dit bestand weggeschreven. Het is zeer belangrijk dat de volgorde van het wegschrijven van de chunks gerespecteerd wordt! Anders krijg je een audiobestand met een foute layout, die bijgevolg niet kan afgespeeld worden.

Alle gegevens worden uit de Data Entities gehaald en onmiddellijk weggeschreven. De SoundData is een uizondering: daar wordt elk deeltje apart weggeschreven.

Using is een alternatieve manier voor Dispose() om een stream goed af te sluiten.

IMG6: writeWav method

# GUI en features

De grafische interface heeft een simpele, maar efficiënte en gemakkelijk te gebruiken structuur die eender wie, zelfs zonder enige voorkennis van zaken, kan gebruiken. Er zijn labels voorzien die uitleg geven hoe de grafische interface werkt. Echter zou het zelfs mogelijk zijn om zonder deze labels het programma te gebruiken.

De gebruiker typt tekst in de textbox, selecteert waar het bestand zal opgeslagen worden en bepaalt de bestandsnaam. Vervolgens klikt deze op de OK button en indien alles correct ingegeven werd, zal de generate button verschijnen. Er zijn gepaste foutmeldingen aanwezig indien de gebruiker iets fout doet.

Om te voorkomen dat de gebruiker alsnog de beveiliging omzeilt door eerst goede info en vervolgens slechte info mee te geven, verdwijnt de generate button en wordt deze vervangen door een label. Het proces moet opnieuw doorlopen worden om een nieuwe audio bestand te genereren.

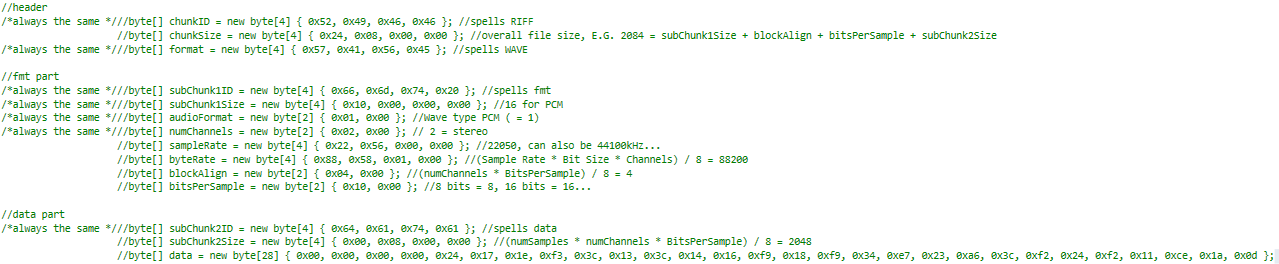
Er zijn 2 zeer handige features ingebouwd die reeds vermeld werden: het kiezen van de bestandslocatie om het bestand op te slagen en het kiezen van de bestandsnaam. Dit zorgt ervoor dat er geen hardgecodeerde bestandslocatie en bestandsnaam in dit programma verwerkt zit. Deze kunnen namelijk voor foutmeldingen zorgen binnen het programma zelf. Zo zal de kans nagenoeg nihil zijn dat er iemand dit programma op een exact dezelfde locatie bewaart als mij.

# Failed experiments

## Problemen aanmaken \*.wav bestand

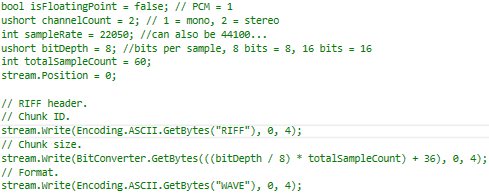
Hieronder vermeld ik de belangrijkste problemen die ik gehad heb en toon ik hoe deze mij tot de juiste oplossing hebben gebracht.

Ik heb eerst geprobeerd om via een hard gecodeerde methode reeds een \*.wav bestand aan te maken. Alle waarden zijn hexadecimale waarden en het data gedeelte bevat willekeurige waarden[[6]](#footnote-6). Echter kon het \*.wav bestand niet afgespeeld worden door diverse redenen: foute data invoer, foute types...



IMG7: failed experiment 1

Onderstaande afbeelding is een snapshot van een stukje code dat ik gebruikt heb. Ik heb letterlijk alles gekopieerd, maar tevens werkte het \*.wav bestand niet. Werken met posities, offset en momory streaming vond ik zodanig verwarrend, dat ik tijdens het debuggen niet wist waar de fout zou kunnen zitten. Bij deze heb ik dit fragment laten varen. Ter controle zag ik wel dat alle formules nagenoeg overeen kwamen waardoor ik zeker wist dat ik die formules moest gebruiken[[7]](#footnote-7).



IMG8: failed experiment 2

Tot slot heb ik alsnog een goed voorbeeld gevonden die ik dan gebruikt heb om een goed werkende \*.wav bestand aan te maken (zie code en sectie 2. e.v.).

## Overige problemen

Ik had het probleem dat men data.SoundData de value null bevatte. Ik maakte een nieuw object van de Data klasse in men GUI. Deze gaf ik aan zowel mijn calculateSamples() als met men writeWav() methode mee. Hierdoor werd niet telkens hetzelfde object bewerkt, maar waren de objecten niet gesynchroniseerd met elkaar. Oplossing: de nieuwe objecten worden nu in de Logica, in de default constructor, aangemaakt.

Om mijn \*.csv file te openen op alle computers, mocht ik mijn path niet hard coderen. Vele errors verder ben ik uiteindelijk genoeg mappen naar boven gegaan via *..\* zodat ik hierop geen error meer kreeg.

# Besluit

Het is niet evident om zo’n programma helemaal vanuit het niets op te bouwen. Ik dacht dat één of twee goede bronnen wel voldoende zou zijn. Daaruit zou ik elke lijn code kopiëren naar de logica klasse en met weinig eigen toegevoegde code het programma doen werken. Dit is helemaal niet zo!

Weten hoe een \*.wav bestand in elkaar zit, hoe het formaat werkt en hoe je alle onderdelen kan berekenen, is zeer belangrijk en goed te begrijpen. Elk stukje code in de bronnen begrijpen en weten wat ze precies doen is dus de sleutel tot een succes verhaal.

Het gemaakte programma loopt soepel en is efficiënt. Het programma doet wat het voor gemaakt is, niet meer, niet minder. Er zouden eventueel wel extra features ingebouwd kunnen worden in de GUI om het programma nog gebruiksvriendelijker te maken. Ik denk maar aan meer characters implementeren, bestandslocatie en bestandsnaam combineren, grafisch mooiere GUI...

# Bibliografie

Craig. (sd). *WAVE PCM soundfile format*. Opgehaald van Soundfile: http://soundfile.sapp.org/doc/WaveFormat/

Johnson, J. I. (2010, November 21). *Writing a Proper Wave File*. Opgehaald van Codeproject: http://www.codeproject.com/Articles/129173/Writing-a-Proper-Wave-File

tomsv. (2013, July 13). *Reading CSV file and storing values into an array*. Opgehaald van Stackoverflow: http://stackoverflow.com/questions/5282999/reading-csv-file-and-storing-values-into-an-array

Topherlee. (sd). *Digital Audio - Creating a WAV (RIFF) file*. Opgehaald van Topherlee: http://www.topherlee.com/software/pcm-tut-wavformat.html

Waters, D. (2009, June 24). *Intro to Audio Programming, Part 3: Synthesizing Simple Wave Audio using C#*. Opgehaald van Msdn: http://blogs.msdn.com/b/dawate/archive/2009/06/24/intro-to-audio-programming-part-3-synthesizing-simple-wave-audio-using-c.aspx

Wikipedia. (2013, December 8). *Sine Wave*. Opgehaald van Wikipedia: https://en.wikipedia.org/wiki/Sine\_wave

Wikipedia. (2015, November 26). *Comma seperated values*. Retrieved from Wikipedia: https://en.wikipedia.org/wiki/Comma-separated\_values

1. Get wordt gebruikt om de value van een property van een klasse te verkrijgen [↑](#footnote-ref-1)
2. Set wordt gebruikt om de value van een property van een klasse in te stellen [↑](#footnote-ref-2)
3. http://blogs.msdn.com/b/dawate/archive/2009/06/24/intro-to-audio-programming-part-3-synthesizing-simple-wave-audio-using-c.aspx [↑](#footnote-ref-3)
4. numSamples = format.SampleRate \* format.NumChannels [↑](#footnote-ref-4)
5. double t = (Math.PI \* 2 \* InputFrequencies[h]) / (format.SampleRate \* format.NumChannels); [↑](#footnote-ref-5)
6. http://soundfile.sapp.org/doc/WaveFormat/ [↑](#footnote-ref-6)
7. http://www.codeproject.com/Articles/129173/Writing-a-Proper-Wave-File [↑](#footnote-ref-7)