**Leander Clappaert**

**2 ICT 4**

**Wetenschappelijke toepassingen 1 – Opdracht 2 – Fourier**

Inhoudstafel

[Inleiding 1](#_Toc438911416)

[Inlezen bestanden 1](#_Toc438911417)

[Het \*.csv bestand 1](#_Toc438911418)

[Het \*.wav bestand 1](#_Toc438911419)

[Verkrijgen van de frequenties 2](#_Toc438911420)

[RFFT 2](#_Toc438911421)

[Converteren naar een float array 2](#_Toc438911422)

[Aanpassen float array tot de gewenste grootte 3](#_Toc438911423)

[De richting van de omzetting 3](#_Toc438911424)

[Verkrijgen van de tekst 3](#_Toc438911425)

[GUI en features 3](#_Toc438911426)

[Failed experiments 4](#_Toc438911427)

[Inlezen \*.wav bestand 4](#_Toc438911428)

[Exception in de FFT methode uit de Fourier klasse 4](#_Toc438911429)

[Foute conversie van mijn data array 5](#_Toc438911430)

[Besluit 5](#_Toc438911431)

[Bibliografie 6](#_Toc438911432)

# Inleiding

Deze opdracht volgt op opdracht 1. Waar we in opdracht 1 tekst naar geluid moesten omzetten en dit vervolgens opslaan in een \*.wav bestand, moet nu net het tegenovergestelde gebeuren. Het gecreeërde \*.wav bestand wordt nu opnieuw omgezet naar tekst. Het is vanzelfsprekend dat de bekomen tekst dezelfde is als de tekst die gebruikt werd om het \*.wav bestand aan te maken.

# Inlezen bestanden

De twee bestanden die nodig zijn voor deze toepassing zijn een \*.wav en een \*.csv bestand[[1]](#footnote-1). In het data gedeelte van het \*.wav bestand zitten de frequenties verborgen die overeenstemmen met een waarde die opgeslagen is in het \*.csv bestand. Door de data frequenties te vergelijken met de frequenties uit het \*.csv bestand kan men te weten komen welke characters er verborgen zitten in het \*.wav bestand.

De bestanden worden niet virtueel opgehaald door de toepassing. De gebruiker kiest de path waar het bestand / de bestanden zich bevinden. Deze path(s) worden opgeslagen binnen de toepassing.

Om er voor te zorgen dat niet elk willekeurig bestand geselecteerd kan worden, is er een eerste controle voorzien. De extensies van de bestanden[[2]](#footnote-2) worden gecontroleerd. Indien een bestand niet de juiste extensie heeft, dan zal de toepassing niet kunnen gestart worden.

## Het \*.csv bestand

Het bestand wordt in de backend van de toepassing ingelezen en opgeslagen binnen een Dictionary object. Het object vertegenwoordigt een collectie van ‘Key’ en ‘Value’ paren. De Keys zijn de characters en de Values de bijhorende frequenties. Dit bestand wordt pas helemaal op het einde van de toepassing gebruikt, maar het is toch reeds belangrijk het bestand in het begin in te lezen. Indien het bestand niet kan worden gelezen, dan zal er een foutmelding gegenereerd worden en zal bijgevolg de toepassing niet starten.

## Het \*.wav bestand

Het bestand wordt tevens in de backend van de toepassing ingelezen. Elk onderdeel van het bestand wordt geanalyseerd en opgeslagen in variabelen. Enkel de bestandsonderdelen die nodig zijn in het verdere vervolg van de toepassing zullen in globale velden worden opgeslagen. De methode dat hiervoor gebruikt wordt is het 1 voor 1 inlezen van die bestandsonderdelen aan de hand van een BinaryReader[[3]](#footnote-3). Een groot voordeel bij deze methode is dat indien het bestand niet kan ingelezen worden, doordat er 1 of meerde bestandsonderdelen fout gecodeerd zijn, er een foutmelding wordt gegenereerd waardoor de toepassing niet zal werken. Een nadeel hierbij is dat er via deze methode ook data ingelezen wordt die niet nodig is voor het voltooien van de toepassing. Echter staat dit toe de toepassing in de toekomst eventueel uit te breiden en zodat er wel gebruik gemaakt wordt van deze data.

# Verkrijgen van de frequenties

De methode GetFrequencies() is de belangrijkste methode van het programma. Daar gebeurt het echte reken- en converteerwerk. De paths van zowel het \*.wav bestand als het \*.csv bestand worden als parameters meegegeven zodat deze uit de backend kunnen worden gehaald via een backend object.

De werking loopt als volgt: de data wordt in kleinere delen opgedeeld. Deze delen (=samples) zijn afhankelijk van de sample rate. Indien de sample rate 44100 Hz is, dan zullen de delen elk zo groot zijn. Hiervan wordt de gezochte frequentie berekend aan de hand van de RFFT methode van Fourier. De bekomen frequentiewaarden zullen niet altijd de perfecte waarden zijn als ervoor, vandaar dat de waarden afgerond worden alvorens ze opgeslagen worden in een integer array.

## RFFT

De techniek die gebruikt wordt om de geluidssignalen om te zetten naar tekst is Fourier, meer bepaald de RFFT (= Reverse Fast Fourier Transform) techniek. Hiermee kan een discreet signaal[[4]](#footnote-4) uitgerekend worden. De statische methode RFFT uit de Fourier klasse, die zich bevindt in de data entities van het project, heeft twee belangrijke voorwaarden opdat deze de data goed kan converteren: de data moet een float[] array zijn met gewenste grootte en de richting van de omzetting moet gegeven zijn. Indien deze 2 voorwaarden voldaan zijn, dan zal de ingevoerde data array aangepast worden.

### Converteren naar een float array

De data uit het \*.wav bestand wordt door de BinaryReader opgeslagen binnen een array van byte waarden. Afhankelijk van het aantal kanalen en de bit depth[[5]](#footnote-5), zal de conversie anders verlopen. Indien er 2 kanalen gebruikt werden om het \*.wav bestand aan te maken (stereo geluid), zal de helft van de data weggefilterd worden in de FilterChannels methode. Dit komt omdat er voor elk kanaal de data beschikbaar is, waardoor je als het ware dubbele data hebt. Dit is niet nodig om deze ook te verwerken.

De bit depth zal bepalen hoeveel bytes er samengevoegd worden tot 1 waarde. Als er 8 bits gebruikt werden, dan zal de data enkel gekopieerd worden. Wanneer er 16 bits werden gebruikt, dan zullen de bytes per 2 worden samengevoegd tot 1 waarde. Bij 32 bits zijn dit 4 bytes per waarde enzovoort. De BitConverter klasse van .NET zorgt ervoor dat deze conversie vlekkeloos verloopt.

Na deze 2 stappen is de nieuwe float array gecreeërd. Merk op: indien het \*.wav bestand gecodeerd werd met de eigenschappen 2 channels met 16 bits per sample, de array met float waarden een kwart zoveel waarden bevat als de originele array. Echter zijn de waarden groter geworden.

### Aanpassen float array tot de gewenste grootte

Opdat de float array zou werken in de RFFT methode moet de grootte van de array kunnen omgezet worden naar een macht van 2 met een natuurlijk getal als exponent. De methode IsPowerOf2 van de Fourier klasse controleert de waarde die meegegeven wordt en returned een true of false indien dit wel of niet een natuurlijk getal als exponent genereerd. Indien het antwoord false is, wordt de AdjustCurrentData methode opgeroepen. Deze haalt de exponentiële waarde op via de Log2 methode van de Fourier klasse om vervolgens af te ronden naar boven. Er wordt een nieuwe array aangemaakt met de grootte 2^nieuweExponent. Alle oude data wordt gekopieerd in de nieuwe array en de de overige plaats wordt opgevuld door 0.

Om dit alles met een voorbeeld te ondersteunen wordt de sample rate van hierboven gebruikt (44100 Hz). 44100 Hz = 2^15,43 Hz. Dit wordt afgerond naar 2^16 Hz = 65536. Een nieuwe array wordt aangemaakt met die grootte. De eerste 44100 waarden zijn dezelfde als de oude array, de overige 21436 (=65536 – 44100) waarden worden opgevuld door 0.

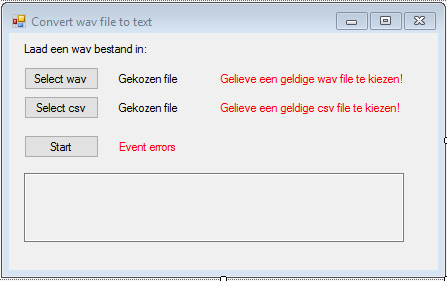
### De richting van de omzetting

Er zijn 2 mogelijke waarden: 1 of -1. Indien 1 ingegeven wordt, dan zal de omzetting van tijds- naar frequentiedomein verlopen (= Forward). Bij -1 verloopt de omzetting van frequentie- naar tijdsdomein (= Backward). Omdat bij deze toepassing de frequenties gezocht worden, gebeurt de omzetting Forward.

# Verkrijgen van de tekst

Nu alle frequenties berekend zijn, kunnen deze vergeleken worden met de vooropgestelde waarden uit het \*.csv bestand. Dit gebeurt in de GetSecretText() methode. Het \*.csv bestand wordt via een backend object ingelezen (zoals reeds uitgelegd in ‘het \*.csv bestand’) en elke frequentiewaarde wordt vergeleken met de degene uit het bestand. De overeenstemmende waarden worden opgeslagen en kunnen vervolgens getoond worden aan de gebruiker.

# GUI en features



IMG1: GUI

De GUI is een zeer basic implementatie van de logica. De gebruiker kiest een \*.wav bestand en een \*.csv bestand vanop zijn computer. Indien het een geldig bestand is, dan wordt het bestandsnaam weergegeven op het scherm. Indien het ongeldig is, wordt een foutmelding weergegeven. De start button is enkel actief indien er geen errors meer zijn.

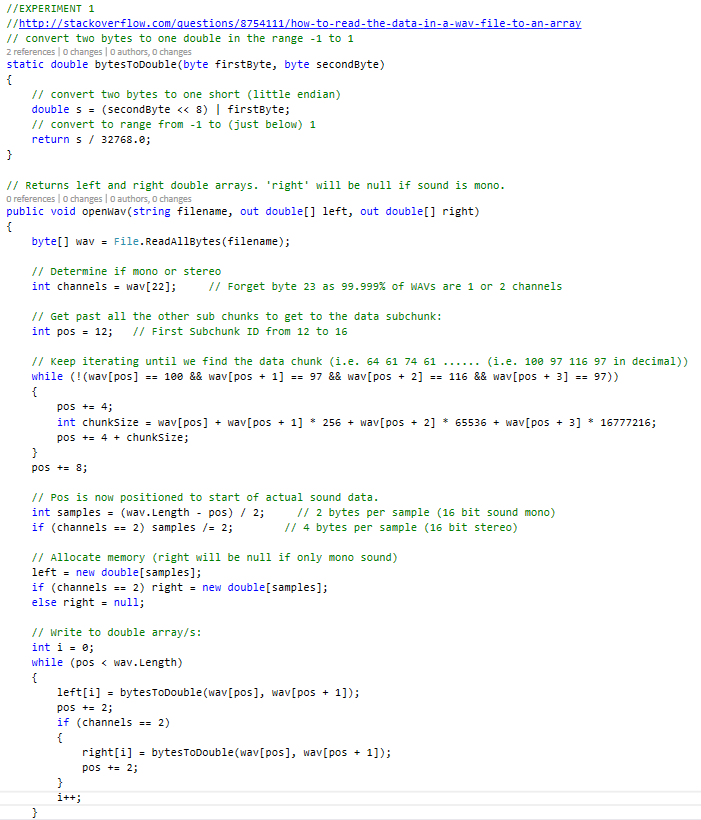
Wanneer de gebruiker op start klikt, dan worden de paths van de \*.wav en \*.csv bestanden opgehaald en meegegeven aan de logica. Enige errors met deze bestanden kunnen worden weergegeven op het programma en zal het programma niet werken. Indien alles goed verloopt en er geen errors te zien zijn, wordt de verborgen tekst in het tekstveld weergegeven.

Nadat de tekst op het scherm komt, of indien er errors waren en het programma niet gestart wordt, dan fungeert diezelfde startknop als resetknop. Hierbij wordt alles opnieuw ingesteld zoals voorheen. Dit zorgt ervoor dat de toepassing niet telkens moet afgesloten worden.

# Failed experiments

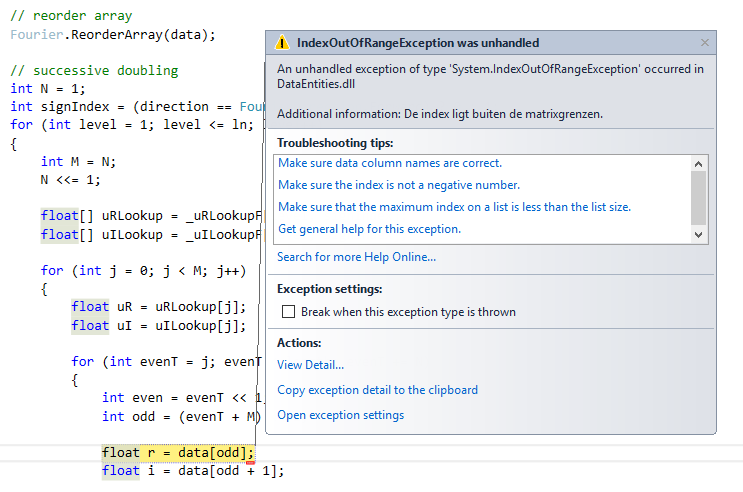
## Inlezen \*.wav bestand

Alvorens ik de methode van msdn[[6]](#footnote-6) gevonden had, gebruikte ik deze methode. Dit was echter niet zo evident om te gebruiken. Je moest al bij andere bronnen gaan zoeken hoe de offset regeling in elkaar zat om te weten op welke plaatsen van de data je bepaalde delen kon uitlezen. Deze methode was enerzijds verwarrend en anderzijds werkte die gewoon niet goed. Er kon te veel foutlopen. Vandaar dat de methode op msdn veel gebruiksvriendelijker is.



IMG2: wrong method used

## Exception in de FFT methode uit de Fourier klasse

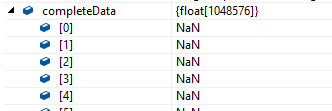


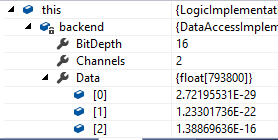
IMG3: IndexOutOfRangeException

Ik gebruikte de FFT methode om mijn data te converteren, echter liep ik tegen een IndexOutOfRangeException die ik niet weg wist te werken. Bij het debuggen van deze methode zag ik dat de ‘odd’ waarde 1048578 was, terwijl mijn data array maar 1048576 groot was.

Dit heb ik opgelost door de RFFT methode te gebruiken, die zich tevens in dezelfde klasse bevindt. Daar genereerde mijn code geen errors.

## Foute conversie van mijn data array





IMG4: data array (bytes) IMG5: NaN data

Het programma leek te werken, echter had ik telkens frequentiewaarde 0. Bij het debuggen bleek het probleem groter te zijn dan gedacht: mijn completeData array, degene waaruit de frequenties moeten gehaald worden, bevatte allemaal NaN’s[[7]](#footnote-7). De originele byte data klopte wel degelijk, dus er na moest er iets gebeurd zijn.

Het probleem was een zeer kleine codeerfout met grote gevolgen. Bij het converteren van de byte array naar een float array werd maar de halve array gevuld met waarden. Hierdoor werd de data na RFFT omgezet naar NaN’s.

# Besluit

Deze opdracht werd sneller voltooid dan de vorige. Dit komt omdat ik reeds weet hoe een \*.wav bestand in elkaar zit, waardoor ik ook weet welke stukjes data er nodig zijn. Het implementeren van de Fourier klasse was toch niet evident omdat je niet weet welke methodes je nodig hebt, alsook hoe die methodes precies werken.

Het belangrijkste is de juiste parameters met de juiste inhoud aan te brengen. Het omzetten naar een geldige float array was even zoeken, maar lukte uiteindelijk toch. Data werd niet altijd goed doorgegeven of werd niet altijd goed geconverteerd. Vandaar dat debuggen enorm belangrijk is.

Ik heb geleerd uit de fouten die ik maakte uit opdracht 1. In de plaats van 2 methodes te gebruiken die elke een array returnde om een \*.csv bestand in te lezen, gebruik ik nu 1 methode en sla de waarden op in een Dictionary. Alsook deed mijn vorige programma nog meer onnodig werk: ik overschreef telkens bepaalde waarden in mijn array. Alsook Had ik onnodige forloops... Nu heb ik geprobeerd het programma zo clean mogelijk te houden. Dit door onder andere bij een stereo \*.wav bestand de helft van de data weg te filteren, omdat deze toch dezelfde is en dus geen invloed heeft op de te bekomen frequentiewaarden.

# Bibliografie

Microsoft. (2015). *Streaming Data from a WAV File*. Retrieved from Microsoft: https://msdn.microsoft.com/en-us/library/ff827591.aspx

R, P. (2011, October 11). *How to get frequency from fft result?* Retrieved from Stackoverflow: http://stackoverflow.com/questions/7674877/how-to-get-frequency-from-fft-result

Yogesh. (2015, December 8). *Reading CSV file and storing values into an array*. Retrieved from Stackoverflow: http://stackoverflow.com/questions/5282999/reading-csv-file-and-storing-values-into-an-array

1. De theoretische achtergrond met betrekking tot een \*.csv bestand en een \*.wav bestand is terug te vinden in het verslag van opdracht 1. [↑](#footnote-ref-1)
2. Een bestandsextensie of kortweg extensie is een toevoeging aan het eind van een bestandsnaam waarmee aangegeven kan worden om wat voor soort bestand het gaat.

   Bron: https://nl.wikipedia.org/wiki/Bestandsextensie [↑](#footnote-ref-2)
3. BinaryReader klasse leest primitieve data types in als binaire waarden in een specifieke encodering.

   Bron: https://msdn.microsoft.com/en-us/library/system.io.binaryreader(v=vs.110).aspx [↑](#footnote-ref-3)
4. Dit is een signaal waarvan de waarden bekend zijn voor een eindig aantal N punten op een eindige afstand van elkaar. Bron: https://nl.wikipedia.org/wiki/Fast\_Fourier\_transform [↑](#footnote-ref-4)
5. Het aantal bits aan informatie waaruit elke sample bestaat. Een andere term is bits per sample.

   Bron: https://en.wikipedia.org/wiki/Audio\_bit\_depth [↑](#footnote-ref-5)
6. Bron: https://msdn.microsoft.com/en-us/library/ff827591.aspx [↑](#footnote-ref-6)
7. NaN = Not a Number. Deze waarde is undefined / unrepresentable een waarde

   Bron: https://en.wikipedia.org/wiki/NaN [↑](#footnote-ref-7)