Auditory Representation of DNA sequence alignments

University of York MRes Bioinformatics
Spring term Project 2003

Katherine Tindall

Supervisor: Dr. Alistair Edwards

Word Count: 7979

Abstract

DNAudio is a program designed to produce an auditory representation of DNA sequence alignments, which are important sources of information for biologists. There are many ways of analysing them, but commonly, the initial analysis is carried out visually. The use of non-speech audio has often been found to be an effective way of representing data, and there are features of sequence alignments that may make them suitable for auditory representation. The aim of this program is to present a user interface by which any user can produce and listen to an auditory representation of DNA sequence alignment data contained within the GDE flat file format. The final versions of the program can represent the data directly (using a one nucleotide = one note sound mapping) or can represent the alignment consensus data (using a different notes for a match column and a mismatch or gap column). User feedback was used to improve the program during design, and the final evaluation was encouraging.

CHAPTER 1: Introduction

Sequence alignment is the identification of residue-residue correspondences. It is *the* basic tool of bioinformatics.
Lesk (2002)[6], pp161, emphasis in the original.

1.1 Alignment and Sonification Theory

With the huge recent growth in biological sequence data (e.g. [1]), it has become essential to find new ways of analysing this data and obtaining information from it.

Sequence alignment is the procedure of comparing two (pair-wise alignment) or more (multiple sequence alignment) sequences by searching for a series of individual characters or character patterns that are in the same order in the sequences. If two biological sequences are similar, this can suggest the sequences have the same function, same structure or a common evolutionary history

Alignment is performed by the following method. Two sequences are aligned by writing them across the page in rows. Characters that are identical, or similar, in both sequences are placed in the same columns and non-identical characters are placed in either the same column as a mismatch or opposite a gap in the other sequence. In an optimal alignment, non-identical characters and gaps are placed so that as many identical characters are in the same column as possible. If two or more sequences can be readily arranged in this way, they are said to be similar.

It is beyond the scope of this project to produce sequence alignments. There are many applications that have this function already (e.g. ClustalX [2], PRRN [3], DIALIGN [4]). Thompson et al. [5] has a good review of these.

It is important for bioinformaticians to find ways of obtaining information from sequence alignments. On a basic level, information can be obtained from sequence alignment data by inspecting the alignment visually. Lesk [6] states, “Visual examination of multiple sequence alignment tables is one of the most profitable activities that a molecular biologist can undertake away from the lab bench”. However, visual processing always has a single focus, therefore it is impossible to study two sequences at once. Auditory processing, however, can be performed in parallel, and therefore may be of use to involve some sort of audio component in the analysis of sequence alignments.

The use of non-speech audio to convey information is called ‘sonification’. More specifically sonification is “the transformation of data relations into perceived relations in an acoustic signal for the purposes of facilitating

communication or interpretation" [7]. There are two basic features of audio perception that suggest that audio display may be effective for representing data in general, and sequence alignments in particular. These are discussed below.

Auditory perception is particularly sensitive to temporal changes, or changes in sounds over time. Human hearing is therefore well designed to discriminate between periodic and aperiodic events [8]. There are features of sequence alignments that may be thought of as periodic. A sequence of DNA codes for a sequence of amino acids (and hence proteins), using a triplet code (three bases of DNA, called a codon, code for one amino acid). Because of the degeneracy of the DNA code (there are 4^3 = 64 codons but only 20 amino acids commonly found in proteins), the third base in each codon is often subject to more mutation than other bases, as mutations of that base are less likely to affect the gene product. This leads to a phenomenon called 'third base wobble', where the first two positions in a triplet are conserved (the same/similar in all sequences) and the third is more likely to be different, and hence a periodicity in coding DNA. It must be remembered that not all DNA codes directly for a gene product, and therefore, in regions that do not code for a gene product, this periodicity would not be expected. Audio display could help in identifying areas of the DNA sequence alignment with periodicity (and therefore hypothetically coding regions) from those without.

Perception of sound does not require the listener to be oriented in a particular direction. Auditory display can be used when the eyes are busy elsewhere. It can therefore be used at the same time as visual display to enhance the amount of information that can be gained in a given period of time from the data available.

Previous areas that audio display has been successfully used include the Geiger counter, where the radiation level data is presented as clicks, allowing the user to have visual concentration elsewhere while continually monitoring the potentially dangerous changes in radiation. More relevantly to sequence alignments, there have been a number of exciting developments in data analysis using sonification. During the Voyager 2 space mission to Saturn there was a problem with the spacecraft, the visual displays only showed noisy signals, but when the data was played through a music synthesiser, a distinctive 'machine gun' sound was heard, leading to the discovery of the problem's cause [7]. Though this is not directly relevant to sequence alignment data, it is true that the auditory system has the ability to extract underlying structure and temporal aspects of complex signals, even when signal noise makes visual processing difficult.

Given the need for biologists to analyse the important data contained within sequence alignments, and the proven ability of the auditory system to be of greater use in identifying data in certain circumstances, it follows that it may be useful to sonify sequence alignments.

The aim of this project is to produce a piece of software that can produce a sonification of DNA sequence alignments. It is hoped that from this sonification users will be able to gain more information than they may be able to from visual data alone. More specifically, the program will be designed to take in a sequence alignment file as input, and produce a sound file with the alignment data encoded within it. In order to make this process 'user-friendly' the program will also have a graphical user interface (GUI). The GUI will also allow the user to play the sound file from within the program.

The sequence alignments to be used are DNA sequence alignments. Protein alignments can also be produced however, there are 20 letters in the amino acid 'alphabet', and only 4 in the DNA 'alphabet'. As the letters will be mapped directly to sounds, the smaller alphabet will be simpler, and therefore the project will concentrate on DNA sequence alignments. If successful, there may be great benefit in studying protein alignments in the same way.

1.2 Similar Work

There have been previous attempts to sonify biological sequence data ([9], [10], [11], [12]). However there has only been one known previous attempt to sonify data in sequence alignments [13]. In this, the consensus between two sequences determines the way in which the sonification is heard. A pair of notes (corresponding to the two sequences) are compared, and a succeeding expression such as "if left input = right input then output X else output Y" assigns two numbers according to similarity/dissimilarity, which are used to produce new values for note-on velocity. Thus, both melodies become louder when domains of high similarity are encountered. Two sequences are recorded into Musical Instrument Digital Interface (MIDI) sequence files and played back in a sequencer program with a display of velocity depicting a similarity profile.

In terms of the sonification, the high level choices that have to be made concern the data that is to be used and the way in which the data is represented. Some previous work on the sonification of biological sequences involved extracting pattern data or motifs from the sequences and encoding them as sound. This allows the user to gain a feel for the structure of the gene or sequence, but makes it more difficult to make a connection between exactly which part of the sequence data is being displayed at any given point. To truly examine the data by audio processing, a direct mapping of the alignment and sequence data is needed, for example where each letter in the sequence corresponds to one sound. This project will aim to produce such a direct mapping.

CHAPTER 2: Project Design

2.1 Languages Used

2.1.1 Program

The primary part of the project is to produce a program with a graphical user interface (GUI). The role of the program is to produce and play a sound file. With these considerations in mind, Perl was chosen as the programming language. The initial task of the program is to obtain data from a text based input file of a set format. Perl is an excellent language for taking data from a text-based file using regular expressions and producing an output in a different format. It also has an extension module (Perl/Tk)[14] for the creation of graphical user interfaces.

2.1.2 Audio Output

Perl is an easily extendable language. Many modules are available for download from the Internet. A module is a collection of subroutines that, when installed, act as built-in functions. Not only does Perl have a module for the production of GUI's, it also has a module that allows MIDI files to be produced. This module: Perl::MIDI [15] has many uses. It can be used for real time sound encoding or a Perl program can be written to encode a MIDI file requiring no audio input. This means that the data in a text-based alignment input file could quickly be transformed, through an intermediate Perl::MIDI program into a MIDI file format, which has data from the original sequence encoded within it.

MIDI has many advantages over other sound file formats in this situation. General MIDI (GM) is a standard for storing compositions based on events that happened during the performance. It does not contain digitised audio data; instead, it stores only information about which notes were played in a time-line format. Therefore, no audio input data is needed, and a MIDI file can be written from scratch by a program. MIDI files are also small (often under 10kb per minute of sound compared to Digital Audio files that may be up to 10MB per minute of sound). This means that files will not take long to load and therefore increase the speed of the application: always a benefit. The disadvantage of MIDI files is that they sound different on different systems depending on the sound card installed. This is because they do not contain audio data: they simply tell the sound card which notes to play. Different sound cards can interpret these commands differently. Therefore in order to make the program portable, care must be taken to ensure that the instructions contained within the MIDI files are suitable for use on all systems.

CHAPTER 3: Objectives and Overview

3.1 Practical Objectives

There is a strict time constraint in place for this project, and therefore a trade off must be made between what is desirable and what is achievable in time available. In the design of the program key objectives were identified that were essential to achieve. Many extension objectives were also identified that would improve the program if time were available.

Essential Objectives

- Create a program that takes an input file and map the data within it to a series of sound events. In the first instance, these sound events are to be a simple representation of each letter in the sequence.
- For these events to be encoded into a sound file.
- To play this sound file by opening an external application from within the program
- To create a GUI to increase the usability of the program.

Extension Objectives

- To investigate possible improvements in the method by which data is mapped to sound.
- To allow the user to have some degree of control about how the sounds are mapped.
- To play the sound file using a player within the program.

3.2 Structure of Program

The final program produced has been named DNAudio. There are, in fact, two versions of DNAudio, the second version being created as a result of the first extended specification discussed above. These two versions encode the data into sound in different ways (they use different sound mappings), but the overall structure of the programs is the same. This section contains a detailed description of this structure. The differences in the two programs are discussed in a later chapter. The choices made in designing the different sound mappings, and the reasons for there being two versions of the final program are described in detail in the next section.

3.2.1 Input

The program takes a GDE flat file format sequence alignment file as input.

Sequence alignments can be produced by a range of alignment programs [5]. These programs can record the alignment in a variety of file formats (e.g CLUSTAL, NBRF/PIR, GCG/MSF, PHYLIP, GDE). These formats vary in the amount of information about the sequences they contain and in the way in which the sequence are represented. The GDE flat file format has been selected as the input format for DNAudio as it is designed to be a simplified format for importing sequence data, and passing it out to analysis functions.

All commonly used alignment packages can also produce these files, therefore maximising the compatibility of DNAudio with the current analysis techniques of as many biologists as possible. GDE flat file format is defined as follows:

- ***type_character short_name***
- ***sequence_data***
- ***sequence_data***
- ***sequence_data***
- ***...***

The type character is # for DNA/RNA, % for protein sequence, @ for mask sequence, and " for text. The short name is dependent on the program that produced the alignment. This is followed by lines of sequence, each ending with a return character. These lines constitute one aligned sequence until the next type character is encountered, or until the end of the file is reached.

Sample Flat File for two *E. coli* tRNAs[16]:

#ECOTRNT4
GGGUCGUUAGCUCAGUUGGUAGAGCAGUUGACUUUUAAUCAAUUGG
NCGCAGGUUCGAAU CCUGCACGACCCACCA

#ECOTRQ1
UGGGGUAUCGCCAAGCGGUAAGGCACCGGUUUUUGAUACCGGCAUU
CCCUGGUUCGAAUC CAGGUACCCCAGCCA

DNAudio can accept text files as input as long as the sequence information is in the same format as a GDE flat file.

3.2.2 Output

The final output of DNAudio is a MIDI file. The advantages of using MIDI format have been discussed above. The output file is by default named <input file name>.mid, however this can be changed using the 'File|Save As...' menu item.

In order to create the MIDI file, DNAudio creates a Perl code file. This is then executed and it is this 'child' file that creates the MIDI file. The Perl file created is by default called <input file name>.out.

3.2.3 Application of DNAudio

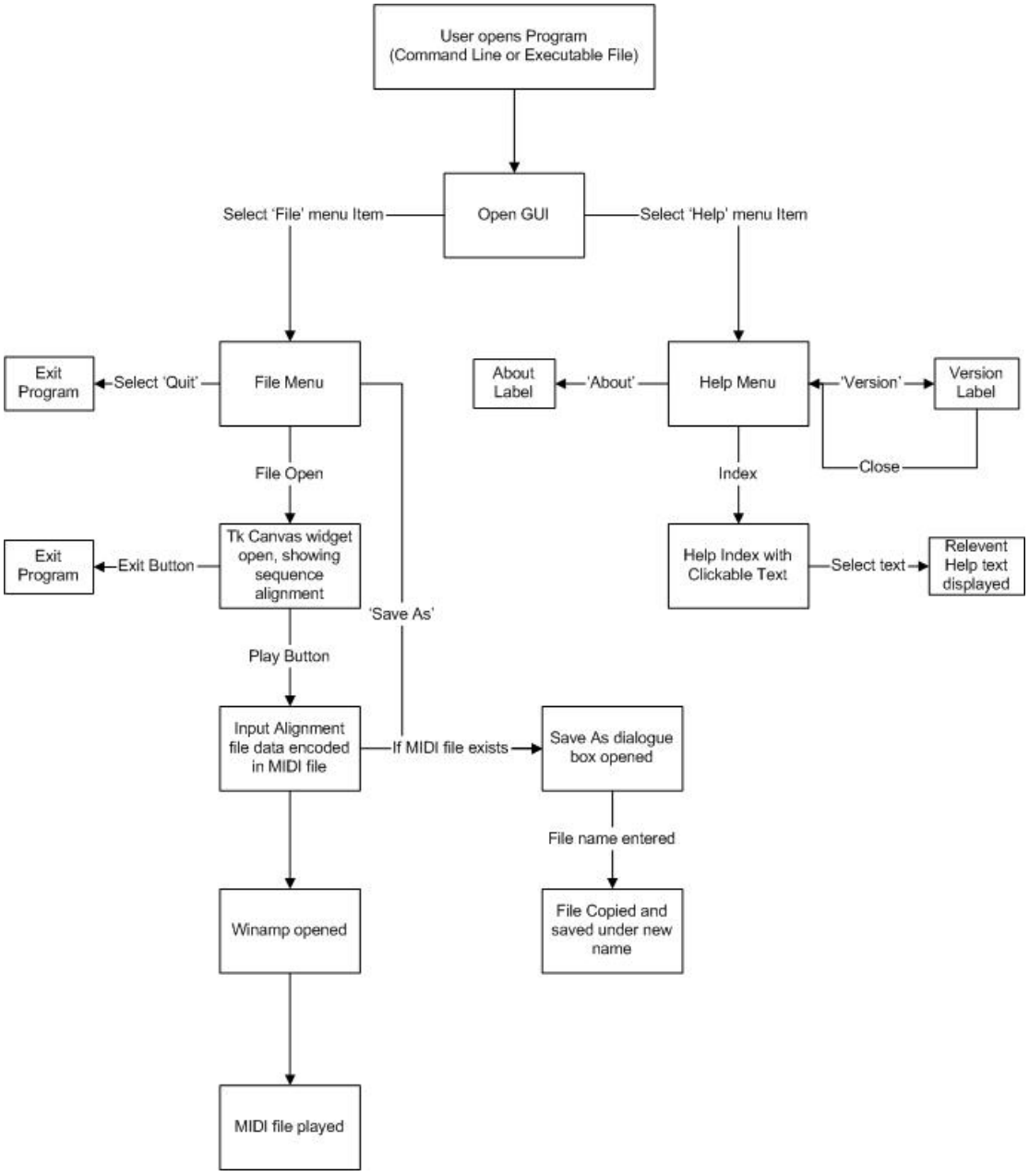

Figure 1. High-level overview of DNAudio.

CHAPTER 4: Implementation

4.1 The main program

The program is executed by running the Perl source code from the command line, or by opening the executable file. As the module Perl/Tk was used extensively, many terms used in the description of the program are based on Tk terminology [17].

With either start method, the program initially produces a Tk MainWindow widget. In a Tk GUI, the MainWindow widget is the first window you create in a program. This is used to contain other widgets, which in turn display what is seen on screen.

Two widgets are automatically displayed when the program opens.

1. A label widget containing an image.
2. A menu bar, with the menu buttons 'File' and 'Help'

Figure 2 The DNAudio Main Window

The image displayed is designed to both provide information to the user and to be decorative. The information it provides is: the name of the program; the

type of input the program is capable of accepting; a brief description of the function of the program and copy write information.

The menu bar is designed so that users can intuitively find commands they need to use in the program. Most non-trivial GUIs have menu bars, and there is a standard convention for their layout, which this program adheres to. There is a 'File' menu icon and a 'Help' menu icon.

The user specifies the file to be opened using the relevant file menu icon, which causes the getOpenFile method to be called. This method opens a dialogue box and prompts the user to select a file to open. By clicking on icons, the user can navigate to and select any file in any directory. The method returns the full path of the file. The program first checks that the information obtained by this method is valid. The input file is opened. If the file cannot be opened, the program will close and present an error message. If the file can be opened, a Canvas widget is opened in the main window.

The Canvas widget displays the sequence alignment, with each base coloured differently. A is displayed red, G, green, C, yellow, and T blue. Each sequence is displayed on a separate line, with the aligned characters directly above and below each other. The Canvas is scrollable, so if the alignment is too long to be displayed in the screen area of the widget, the user will be easily able to access the rest of the data.

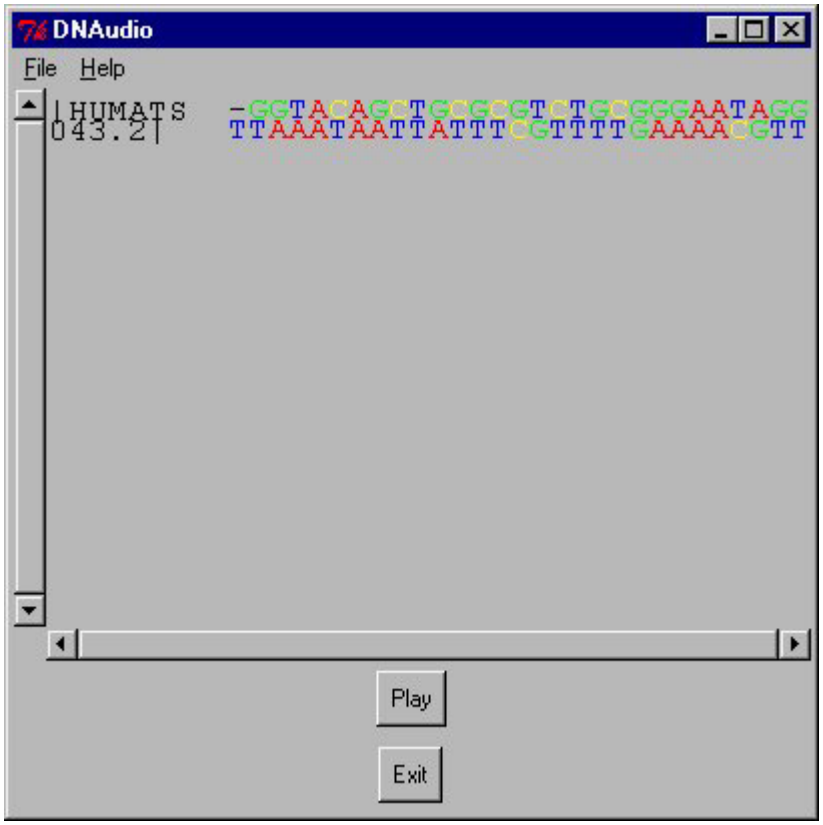

Figure 3 DNAudio displaying a sequence alignment

The text is added to the canvas widget using the createText method. However, before this is used, information is extracted from the input file and stored in a number of variables. Each line of the file is read, and if it begins with a # (signifying a DNA/RNA sequence name, in the context of this program, the only types of sequences used), the line is stored in the @names array. If the line does not begin with a #, it is assumed to be a sequence line, and is stored in the @sequence array. A 'for' loop is then used to go through each sequence name and prints these into the canvas at the same x coordinate position on successive lines. The corresponding element of the sequence array is then itself split into an array (@thissequence). Each element of this array is examined using another loop. 'If' statements print each element to the canvas in the correct colour, dependent of which base they represent. Finally, the input file is closed.

As the sequence alignment is basically a text file, it can be displayed using a text widget. However, it was important to display the file with different colours for each of the bases, and in order to change the colour of text in a Text widget, the text must be tagged. Tags are assigned using the position of the text (e.g. a tag can be positioned by labelling a particular character of a particular line), but because any suitable file can be opened in DNAudio, it is not possible to know in advance exactly where each base type lies.

Below the canvas widget, two further widgets are placed in the main window. Both are "Button" widgets. One is labelled 'Exit' and this closes the program when selected. The other is labelled 'Play'. When this is clicked by the user, it:

- produces another Perl program (in which the code for creating a MIDI file is contained);
- runs this Perl program (thus creating the MIDI file); and finally opens the Winamp playback device, which plays the MIDI file.

In more detail, the subroutine called when the 'Play' button is pressed initially reopens the input file, closing the program and signalling an error if this is not possible. Secondly, an output file is also opened, again with an error handling process that closes the program and reports the error. This output file is where the child Perl program will be written.

Once the required files are opened and have been assigned file handles, a 'Foreach' loop goes through all the lines of the file. As the names of the sequences are not important for the sonification of the sequences, they are not processed in any way, and if the line begins with a #, the loop is exited.

On the sequence lines (those that do not begin with a #), the program first converts all the characters to upper case. The sequence is then split into an array (called @array).

The remaining part of the loop is concerned with the assigning of note values to the various features of the alignment file. The reasons why the note values are chosen are described in the next section.

It was decided that it would be useful to mark splice sites with a separate, easily identifiable, sound. The reasons for this decision are described in the next section. Splice sites are most commonly marked with a 'GT' base sequence. The program is therefore required to identify these sites. To do this a separate array of notes is held that contains note information for only splice sequences.
A 'For' loop goes through the sequence array and where there is a G followed by a T, a 'note' element is inserted into the splice score array. If the base letter is not a G, or it is a G not followed by a T, a 'gap' element is inserted into the splice score array.

At this point it is required to specify a sound mapping from base to sound sequences. Initially a direct mapping was used, each base being directly mapped to a particular sound. The reasons behind the decision to use this method are discussed in the following chapter. Following code testing a second mapping method was also created. This will also be discussed in full in the next section.

The process each sound mapping goes through involves a loop structure which, like the one that produces the visual representation of the sequences, looks at the first element of a line and then, if the line is a sequence line (i.e. it does not begin with a #), the line is split into an array. The reasons for having the similar loop processes separate are:

1. It makes altering a single part (i.e. the display or the sound file production) easier. This may be desired by a subsequent developer, or in the context of this project, after the user evaluation is carried out, there may be improvements that can be made. Separation of the graphic display section from the sound file production section allows alterations to be easily made to each one without affecting the other, and for quicker identification of the relevant code.
2. It means that the child Perl file and the sound file are only produced when the play button is pressed. They could be produced when the program initially opens the input file, but if the user decides that a sonification of the data is not needed after visual analysis of the sequences, they will simply take up memory and be of little use.

4.2 The child Perl file

Once the input file has been processed and arrays have been assigned the correct elements, the child Perl file is created. This is done by printing the necessary code to the already opened output file. The program has already created arrays that contain the note information to be put into the MIDI file. By processing each element in each sequence array, a new element is added to the 'splice_score' array. This element contains the note information for the splice sites in one sequence. The score array contains the information from the main sound mapping. Where there is a direct mapping of the bases, a new element of score array is created for each of the sequences. When there is mapping only of the similarities and differences, there will only be one element of the score array. In either case, for each element of the 'score' array and the 'splice_score' array, a MIDI track is created, using the method

MIDI::Track->new. These tracks are given the name track<track number>. The number of tracks created is flexible and dependent on the sequence file opened. A track is a collection of events that get encoded in to the MIDI file.

Each track contains the contents of one element of @score or @splice_score, i.e. the sonification of the data contained within one sequence.

Once the tracks have been written, they are written into an Opus. MIDI::Opus provides a constructor and methods for objects representing a MIDI opus (also known as a "song"). An opus object has three attributes: a format (0 for MIDI Format 0), a tick parameter and a list of tracks objects that are the real content of that opus. The opus is then written into a file. This is the code that actually creates the MIDI file.

The child Perl file is executed by using the “system” command. The system command is the simplest way to launch a child process in Perl. The disadvantage of this is that the parent program waits for the child process to finish before proceeding. This can be a problem, but here the time taken for the program to run is minimal, so the simplest method has been chosen.

4.3 Creating and playing the MIDI file

Once the child program has been run, the MIDI file is created and in the current directory. Therefore, to play the file, the program can simply use another system command to launch Winamp from the command line, with the name of the created MIDI file as an argument. However, if this method is chosen, Winamp will have the user input focus, and therefore other features of DNAudio will not be available while the audio file is playing. The user will not be able to access the help menu, and, more importantly, will not be able to scroll along the sequence. Therefore, Winamp is launched using file handle. The syntax for launching a parallel child process (i.e. Winamp and DNAudio running concurrently) is to put the command as the “filename” for an open call, and (in this case) precede the command with a vertical bar (the pipe character).

Once Winamp is opened and playing the file (the file plays automatically on opening), the playback can be controlled by the user via the controls built into Winamp (e.g. stop, pause, etc.).

The reason for choosing to open an external device to play the file was one of time. An integrated playback device would be a great addition to the program, but was not achievable within the time available. The primary object of the project was to achieve sonification of sequence alignments and analyse the way in which people could get information from the sonified data. The GUI of the program was simply a means to an end, and thus the outlay of time required for developing an integrated playback device could not be warranted.

Finally, another feature of the main program, not previously described, is the ‘File|Save As…’ menu item. Though this is created at the initial execution of the program, it has no purpose until the MIDI file has been created, and thus

is described at this point. When the menu item is selected, a dialogue box is opened using the getSaveFile method. This has similar properties to the getOpenFile dialogue box. However, the full path of the file name that is entered by the user is used in a copy command at the command line (again using “system” to access the command line). The MIDI file that has previously been stored as the default file name <input file>.mid is now also stored as whatever the user prescribed in the ‘Save As’ dialogue box. The file stored under the old file name can be deleted by the user if they chose to do so.

CHAPTER 5: Choice of Sound Mapping

5.1 Direct Mappings of Sequences into Sound

A sequence alignment can provide information on the similarity of two sequences, and this information can be used to infer functional, structural and evolutionary information about the two sequences. Visual representation of the sequences shows where there is similarity because aligned bases are in the same vertical column. The method of sonification of the sequence alignment must also give the user information about where the sequences in the alignment are the same and where they are different (either where there are two different bases, or a base in one sequence corresponding to a gap in the other sequence).

This can be represented in many different ways. It was an aim of this project to have a direct mapping of the sequences into sounds, thereby not losing any of the information contained within the sequences or in their alignment. Therefore, it was decided that there should be four sounds, each representing one of the four bases, and a rest where there is a gap in a sequence. This is closely analogous to the visual representation, where four differently coloured letters are used to represent the four bases, with a gap usually represented by a – or . symbol. It was hoped that using this mapping would be beneficial for the user who has previous experience in visually analysing sequence alignments. The mapping is simple to explain and understand, effectively the same information is being supplied, but it is being sensed in a different way.

In practice, this sound mapping means that when two aligned sequences have the same base in the same position, two identical (and therefore consonant) sounds will be played simultaneously. When a base is aligned with a gap, only one sound will be playing (in a pairwise alignment). Most people can perceive the difference between a single note playing and two consonant notes playing. When two notes are played simultaneously, the user hears a richer sound quality than when a single note is playing. When two different bases are aligned, the two notes played will not be consonant, and therefore the user should be able to tell that there is a mismatch.

It was hoped that the auditory representation of a sequence alignment might be of significant use to the user because when analysing alignments visually, focus can only be held on one sequence at a time. To be completely accurate, the user must look at each base in a vertical column and judge whether there is a consensus of the bases, or whether there is a mismatch of the bases. To do this quickly is difficult (though most alignment programs also show a representation of regions of consensus, by placing an asterisk in the consensus columns, or by showing a histogram of the level of consensus in each column, for example). When a different pitch of note represents each base, the user can concentrate on identifying the sections where there are dissonant (i.e. discordant, inharmonious or ‘jarring’) sounds, as these will represent mismatches.

Once the basics of the sound mapping had been decided upon, the next choice was the exact pitches of the notes to use. The most important decision was to decide the intervals between the notes used for the different bases.

The sonification of the sequence alignment is designed so that the user can identify where there are mismatches between the sequences in the alignment. Mismatches in homologous sites of an alignment represent evolutionary events. These evolutionary events are often substitutions of bases, but can also be insertion or deletion events (indels). There are, however, different classifications of substitutions, and it was decided that the sound mapping should take these factors into account. Nucleotides are either pyrimidines (Thymine and Cytosine) or purines (Adenine and Guanine). Substitutions that exchange a purine for another purine, or a pyrimidine for another pyrimidine are called transitions, and in some genes are more common than those that exchange a pyrimidine for a purine or vice versa: transversions. The note values chosen to represent each base reflect this. The note intervals between the two purines and between the two pyrimidines were chosen to be more consonant than the intervals between any purine and any pyrimidine.

The simplest approach to quantifying consonance is to say that two tones are consonant if their frequencies are related by a small integer ratio. The ratio determines the musical interval. For example, the octave 2:1, fifth 3:2, and fourth 4:3 are thought to be universally consonant musical intervals.

The sound mapping was chosen to be:

Base	Note
A	B
G	F#
C	C
T	G

Using this sound mapping it was possible to use DNAudio to produce sonifications of DNA sequence alignments. This was the first aim in the project. However, it is possible to find other information about genes from their DNA sequences, and further development of DNAudio could help obtain this information.

This system of sound mapping was fully implemented in the program, and tested using short sample sequence alignments. It was important that the final program should be useful as possible, so before a full user evaluation took place, a short user evaluation was carried out, using sound files created by the program to find what instrumental sounds users found the easiest to gain information from, and which was most pleasing to listen too. The full results of this are shown in the results section. On the basis of these results, the instrument ‘Acoustic Guitar’ was chosen.

5.2 Easing Identification of Splice Sites

In Eukaryotic organisms (those with a cell nucleus), the genetic material contains introns – regions of DNA that are not used for protein synthesis, but are instead discarded ('spliced') from the coding regions of genes during protein synthesis. The remaining pieces of DNA, which together encode the finished protein, are called exons. Because introns are removed from genes, the DNA sequences within them are subject to more mutation than within exons. Substituting bases within introns will not cause a change in the final gene product, but a substitution within an exon may do, and thus exon regions are more conserved. With the sound mapping used in DNAudio, therefore, there should be fewer dissonant sounds in an exon, compared to an intron. This may allow the user to identify putative exons. Furthermore, there is also a strong conservation of sequence around the intron-exon boundaries in eukaryotes: introns nearly always begin and end with the bases GT and AG respectively. Therefore, if a note sounded when there was a GT sequence, it could further help the user identify coding and non-coding regions of the genetic code.

Initially, it was hoped that the splice site sounds could be encoded in the sound file using a different instrument sound from the normal base notes. However, the method in which the MIDI file is encoded meant that only one instrument (or patch) could be playing at one time. It was therefore decided that the 'GT' event notes should be at a pitch more than an octave higher than the other notes, so that the two types of information will not be confused.

5.3 Improving Sound Mapping

Once the above design choices had been implemented, a full user evaluation was designed. This was to use three 'types' of genes. One, an alignment of two "full" gene sequences, containing introns, exons, promoter regions and tail regions, one of the coding regions of two closely related genes (e.g. the same gene from two strains of the same species) and one of the coding regions of two distantly related genes (e.g. homologues from different domains). However, when the full gene sequence was entered in the program, the resulting sonification was 27 minutes long. This length was both unfeasible for the user evaluation and unworkable in practice. The initial alteration that was made to improve the program was to decrease the length of the notes, thus reducing the time taken by the full gene sequence sonification to 5 minutes. However, this was too quick to obtain information from and therefore a new method of sound mapping had to be devised that allowed the information within the alignment to be processed more quickly.

This was achieved by comparing the two aligned sequences, and mapping a higher note where the bases are identical and a lower note where the bases are mismatched or where there is a gap in one of the sequences. The 27 minute long gene could be represented in 6 minutes, while still communicating information about the alignment. Though this gave a great benefit, it also restricted the program, because the mapping only allows pairwise alignments (alignments of two sequences). The original mapping (within the constraints

of the MIDI format) allowed up to 8 sequences to be represented. As the second sound mapping was designed to allow the users to process the information quickly, and the information is gained from whether the note is high or low, rather than the ‘quality’ of the note or any discord, it was also decided that the Glockenspiel sound should be used in this program. If there had been sufficient time, it would have been appropriate to carry out anther user survey to provide evidence of the validity of this decision.

As, for short sequences, the first mapping produces sound files of a reasonable length, and could give more information, it was decided that both mappings could be of use to a user, and could be exploited in different ways. It was decided that the user evaluation should allow the user to use both versions of the program.

CHAPTER 6: User evaluation results

User Evaluation 1

Method

The user was asked to listen to 2 sound files produced by DNAudio1 from an artificial sequence alignment designed to contain all the possible features of a real sequence alignment in a short time period. The user was not told what the sonifications represented, nor were they asked to give any interpretation of the data. For each file they were asked to comment on how easy it was to identify the different features of the sonification (where notes were in unison, where one note was playing solo, where there was an unusually high note, where there was a gap, and where there was discord). Their response, along with their musical background was recorded. One file used the Glockenspiel instrument sound for the notes, and the other the Acoustic guitar.

User	1	2	3	4	5
Preference	Guitar	Guitar	Glockenspiel	Guitar	Guitar
Comment	Notes more sustained			Longer notes	

Table 1: Results of User Evaluation 1

These results suggested that the acoustic guitar sound should be used. The users reacted most favourably to this as it was felt that the longer sustain of the note allowed them more time to get information from it (especially useful when identifying discord).

User Evaluation 2

User Evaluation 2 was a more in depth study of the performance of all aspects of the program. Again, the musical background of the participant was recorded. DNAudio is designed for biologists to use, however, the ability of the user to obtain information from sound may be affected by their musical aptitude. Their biological background was also recorded, especially their previous experience with sequence alignments. Someone who has never encountered a sequence alignment may be expected to be slower at obtaining information from one than an experience molecular biologist, for example.

Each user was then asked to record the information they could obtain from two sequence alignments. Each sequence alignment was presented visually and then with both sound and vision, making four trials for each user in total. The DNA sequences used for the trials were chosen to reflect the perceived strengths of DNAudio1 and DNAudio2.

DNAudio1 used a short alignment of two closely related genes, with no gaps and few mismatched bases, specifically: the partial coding sequences of Pirital virus strain 3673 nucleoprotein (N) gene, and Pirital virus strain 3367 N gene. Users were asked to find the number of mismatched bases in the alignment, first by vision alone (for which they had a 5 minute time limit) and then using the auditory representation provided by DNAudio1.

DNAudio2 used the complete coding sequences of *Homo sapiens* alanyl-tRNA synthetase, and *Caenorhabditis elegans* alanyl-tRNA synthetase. These are reasonably distantly related organisms, and therefore the alignment contained many mismatches and gaps. Users were asked to identify where there are two identical bases in the two sequences, first by sight then using the auditory representation.

Finally, users were asked open-ended questions to allow them to fully express their opinion on the program. They were asked to comment on the ease of use of the program, which method of data presentation preferred for each alignment and what improvements could be made to the program.

The full results are contained in appendix 2.

Summary of results

DNAudio 1

	Number of mismatches	Average Detected	Range of responses
Vision	34	33.181818	30-34
Sound	34	34.181818	30-41

Table 2: Summary of user evaluation of DNAudio1

DNAudio 2

	Number of matches	Average Detected	Range of responses
Vision	1259	309.36364	84-630
Sound	1259	1185.2727	879-1311

Table 3: Summary of user evaluation of DNAudio2

The most notable of the results is that users appeared to find it considerably easier to acquire information from the sonification of longer sequence alignments using DNAudio2 than by vision alone. None of the participants managed to visually analyse the whole of the second sequence alignment in the time limit, and all got a much closer estimate of the number of matched pairs using the sonification.

The results are less clear in the case of DNAudio1. The mean score of all users in the audio results is closer to the actual number of mismatches between the two sequences, than the vision only scores. However, the range of responses is larger in the audio test, and only one user out of eleven identified the correct number of mismatches using the sonification, compared to 6/11 when using vision.

When asked to state which method of analysis they preferred, all users suggested that for shorter sequence alignments, it was easier to identify

mismatches by sight, whereas for longer sequence alignment it was easier to identify matches using the sonification.

CHAPTER 7: Conclusion

In summary, this project has produced two fully operational programs that produce an auditory representation of DNA sequence alignments. The two programs have the same user interface, but differ in the way data is mapped into sound. These two sound mappings allow the user to chose the level of detail at which to study the alignment. DNAudio1 has a more 'direct' sound mapping, where each base is encoded as a different note, and therefore in a sequence alignment, two or more streams of notes are played concurrently. Where there are mismatches in the sequences the auditory representation will have a clash of two different notes. The level of dissonance between the two notes is dependent on the likelihood of substitution of the two bases.
DNAudio2 has a higher-level sound mapping, in that the sonification is a representation of consensus data inferred from the original sequence alignment. This allows the user to process a longer alignment more quickly than the original mapping allowed. Both programs also analysed the data, and where a hypothetical splice site signal (GT) was encountered a higher pitch note was played.

The primary motivation for creating these programs was to see if users can benefit from using an auditory representation of DNA sequence alignments, compared to the commonly used visual analysis. These benefits may be in terms of increasing the amount of information that the user can get out of the data, or decreasing the amount of time that is taken to analyse the data, for example. DNAudio2 appears to be successful in allowing users to analyse long sequence alignments in a shorter time period than visual analysis, whilst still providing accurate information on the comparisons between the two sequences. DNAudio1 appears to be less successful. With short sequences it is possible to analyse the data accurately and quickly by eye, therefore an auditory representation presents little benefit to the user.

The splice site signals were not successful in either of the two programs. Users often felt that these signals got in the way of their interpretation of the other data. Based on the user feedback, the DNAudio program has been altered. The splice site notes are no longer added to the auditory representation, and the lengths of the notes in DNAudio2 are now longer. When the program was being designed, it was thought that users would want to analyse an alignment in the most efficient way possible, and therefore the note length was chosen to be the shortest that would still be able to convey all the necessary information. However, users felt that a longer length of note would allow them to be surer of the information they were inferring from the data, and therefore, though the alignment would have a longer sonification length, the information gain by the user would be more efficient.

In terms of the user interface, all users found its use reasonably straightforward and intuitive. It is based on the standard system used by most modern applications so this is to be expected. However, small design features such as the scroll bars could be improved. Currently there is no indication of the position of the viewing area in terms of the whole sequence. Adding this feature would be an improvement. The program could also be

improved by integrating the playback device with the main program. This would be aesthetically more pleasing; it would make portability of the program easier (as the user would not need to have Winamp installed and available in the same directory as DNAudio); and it would improve the speed of the application – currently there is a long pause between the user pressing play in the main program and the start of the playback due to the time Winamp takes to open.

In terms of the sonification, more user testing could be done to get more feedback on the needs of the user. The project as a whole may have benefited from an initial period of surveying potential users and finding out their needs and preferences before the program was designed. It has been shown that it is possible to encode the data into sound by direct mapping, higher level mapping and also to infer different types of information from the data that can in turn be encoded as sound. For the user to be able to select what they want represented in the sonification would be a good improvement to the program. At the moment, the user can simply run the version of the program they want to use, however, an integration of the two programs would make it more flexible. There are also more features of gene sequences and sequence alignments that have not been considered here at all. Though users suggested that having the splice site sounds distracted them from the main data, they may find that once they are familiar with the data they may want this extra layer of information. Further improvement of the program could allow the user to select which features they would like represented in the sound file. The range of features could be expanded to include, for example, the TATA box, ribosome binding sites, or polyadenylation signals.

DNAudio is a successful attempt to produce an auditory representation of DNA sequence alignments, but many improvements are still possible.

Appendix 1: User Manual

System Requirements

In order to be fully functional, a Perl compiler (Active Perl [18] is the most common for Win32 systems) must be in the path of wherever the program is run. This Perl compiler must have Perl::MIDI installed (if the program is run from the executable file). Perl/Tk must also be installed if the program is to be run from the source code.

The play back device "Winamp 3" [19] must be installed and in the current directory. Winamp is a freeware device for the playback of many media file types, including MIDI.

Installing Perl for Windows

Go to www.activestate.com, and select 'Downloads'.
Select the ActivePerl download and follow the instructions on screen.

Installing Perl::MIDI and Perl/Tk

Once ActivePerl is installed, open the MSDos prompt and change directory to wherever Perl is installed.
Type 'ppm' to open the Perl package manager.
Type 'search' Tk.
A list of the currently available Tk modules should appear.
Type 'install <most recent Tk bundle name>'.
Follow the instructions on screen.

Once Tk is installed, use the Perl package manager to search for 'Perl::MIDI'.
Type 'Install <most recent Perl::MIDI file name>'.
Follow the instructions on screen.

Exit the Perl Package manager.

Installing Winamp

Go to www.winamp.com and click on 'download Winamp3'.
Follow the instructions on screen.

Using the program

Running the Program

The program can be run from the command line, by typing
Once the program has opened, displaying an image and a menu bar, select "File|Open…" from the menu bar to open the sequence data file.

The sequence alignment will be displayed in the window. The scrollbars can be used to move along the sequence.

To hear the sonification of the sequence alignment, press the play button. This will create a MIDI file and will open Winamp to play the file, provided that Winamp is correctly installed in the current directory. The MIDI file will

automatically be stored under the file name <input file>.mid. Select File|Save As…, to store under a different file name (after the file has been created).

The playback of the MIDI file can be controlled using the controls in Winamp.

For more information on the program, click on the Help menu. The Help index contains information on the input, output and display of the program. Select “Index” from the Help menu, and click on the relevant index item for further information.

The program can be closed by selecting File|Quit, by pressing the close button that appears when the sequence file is displayed on the screen, or by clicking the X icon in the top right corner of the window.

Appendix 2: User evaluation Data

User Number		1	2	3
Musical Background	Qualifications	Grade 1	Grade 7	AS level
	Listening Habits	High	High	Medium
	Personal Rating	Low-Medium	Medium	Medium
Biological Background	Education	Degree Level	Degree Level	Degree Level
	Sequence Alignment	Yes	Yes	Yes
Features identified				
DNAudio1	Vision	34	34	30
	Sound	41	32	47
Comments				
DNAudio2	Vision	630	303	84
	Sound	1300	1197	1050
Comments			Periodicity and areas of lower numbers of similarities	
Ease of Use		Good	Good	Quite Good
Preferred Method	DNAudio1	Vision	Vision	Vision
	DNAudio2	Sound	Sound	Sound
Improvements		Better playback interface	Longer notes in DNAudio2 (slower)	Scroll Bar showing position

User Number		4	5	6
Musical Background	Qualifications	Grade 2	Grade 1	~
	Listening Habits	High	High	Low
	Personal Rating	Medium	Low-Medium	Low
Biological Background	Education	Degree Level	GCSE	A-level
	Sequence Alignment	Yes	No	No
Features identified				
DNAudio1	Vision	33	34	33
	Sound	34	31	31
Comments				
DNAudio2	Vision	290	327	213
	Sound	1258	1311	879
Comments			Areas where there were fewer similarities	
Ease of Use		Quite Good	Good	Good
Preferred Method	DNAudio1	Vision	Vision	Vision
	DNAudio2	Sound	Sound	Sound
Improvements		No splice sounds. DNAudio2 slower	Scroll Bar showing position	

User Number		7	8	9
Musical Background	Qualifications	Grade 4	~	Grade 1
	Listening Habits	Medium	Medium	Medium
	Personal Rating	Medium	Medium	Medium
Biological Background	Education	GCSE	Degree Level	Degree Level
	Sequence Alignment	No	Yes	Yes
Features identified				
DNAudio1	Vision	32	34	34
	Sound	37	31	30
Comments				
DNAudio2	Vision	176	263	311
	Sound	1217	1321	1207
Comments				
Ease of Use		Good	Good	Quite Good
Preferred Method	DNAudio1	Vision	Vision	Vision
	DNAudio2	Sound	Sound	Sound
Improvements		DNAudio2 slower		

User Number		10	11
Musical Background	Qualifications	~	Grade 3
	Listening Habits	High	High
	Personal Rating	Medium	Medium
Biological Background	Education	School	Degree Level
	Sequence Alignment	Yes	Yes
Features identified			
DNAudio1	Vision	34	33
	Sound	30	32
Comments			
DNAudio2	Vision	317	489
	Sound	1097	1201
Comments			
Ease of Use		Good	Good
Preferred Method	DNAudio1	Vision	Vision
	DNAudio2	Sound	Sound
Improvements		DNAudio2 slower	

Appendix 2: Full Code for DNAudio

```
use Tk;
use Tk::Entry;
use Tk::Photo;
use Tk::Menu;
use Tk::Canvas;
use Tk::Scrollbar;
use Tk::Text;

########################
# CREATES WINDOW
########################
my $mw = MainWindow->new;

$mw->title("DNAudio");

########################
# PLACES IMAGE IN WINDOW
########################
$image = $mw->Photo(-file => "front.bmp");
my $label = $mw->Label (-image => $image)->pack (-side =>
'top', -anchor => 'w');

########################
# CREATES MENU BAR
########################
$mw->configure(-menu => my $menubar = $mw->Menu);

my $file = $menubar->cascade(-label => '~File');
my $help = $menubar->cascade(-label => '~Help');

######################
# FILE MENU
######################
my $types = [
['GDE Files', ['.gde']],
['Text Files', ['.txt', '.text']],
['All Files',   '*',   ],
];
$file->command(
              -label => 'Open',
              -accelerator => 'Ctrl-o',
              -underline => 0,
              -command    =>    sub    {$file2    =    $mw-
>getOpenFile(-filetypes => $types ); &display ($file2) if
defined $file2;}
              );

$file->separator;
my $types2 = [
['MIDI Files', ['.mid']],
```

```
['Text Files', ['.txt', '.text']],
['All Files',   '*',   ],
];
$file->command(
              -label => 'Save As...',
              -accelerator => 'Ctrl-a',
              -underline => 1,
              -command   =>   sub   {$savefile   =   $mw-
>getSaveFile(-filetypes => $types2, -defaultextension =>
'.mid' ); &savemidi ($savefile) if defined $savefile;}
              );
$file->separator;
#$file->command(
#              -label => 'Close',
#              -accelerator => 'Ctrl-w',
#              -underline => 0,
#              );
#$file->separator;
$file->command(
              -label => 'Quit',
              -accelerator => 'Ctrl-q',
              -underline => 0,
              -command => \&exit,
              );

######################
# HELP MENU
######################
$help->command(-label   =>   'Index',   -command   =>   sub
{&helpwidget});
$help->separator;
$help->command(-label   =>   'Version',   -command   =>   sub
{$answer => $mw->messageBox(-title => 'DNAudio',

-message => 'Version 1.0',

-type => 'Ok'); });
$help->separator;
$help->command(-label   =>   'About',   -command   =>   sub
{$answer2 => $mw->messageBox(-title => 'DNAudio',

-message => "DNAudio is an application\nfor producing an
Auditory     representation\nof     Multiple     Sequence
Alignments\n\n\nUniversity  of  York\nMRes  Bioinformatics
Spring Project 2003\nKatie Tindall",

-type => 'Ok');});


MainLoop;
```

```
########################################
#                                      #
# Now the sub routines                 #
#                                      #
########################################

######################
# DISPLAY
#destroys picture and opens canvas and play buttons
#when pressed, the play button creates a midi file, and
#opens winamp to play it
######################
sub display {

$label->destroy;
$image->delete;

open (FH, $file2) or die "could not open file";
open (OUTPUT, ">$file2.out") or die "Cannot create
file\n";

$canvas = $mw->Scrolled('Canvas')->pack();

#Look through every line

foreach(<FH>) {

              $seq = $_;
              if ($seq = /^#/)  {
                                #add names to array
                                push (@name, $_);
                                }
              else              {
                                chomp;
                                #add sequences to array
                                push (@sequence, $_);
                                }
              }
              $number = @name;
              $rnumber = $number-1;
              $length = @sequence;
              $rlength = $length-1;

              foreach (0..$rnumber){
                                #split each sequence
int an array
                                $poo = $_;
                                my $id = $canvas-
>createText(30,($poo*10)+20, -text => "$name[$_]", -font
=> "courier");
```

```
                                        $sequencesp          =
$sequence[$_];
                                        @thisseq =  split //,
$sequencesp;
                                        print              OUT
$thisseq[1];
                                        $h = 0;

              foreach (@thisseq){
                                    #print  each  letter  or
each sequence array in window
                                    $h++;
                                    if ($_ eq 'A' or $_ eq
'a') {
                                                my  $iq  =
$canvas->createText(($h*10)+200,(($poo*10)+10),  -text  =>
"A", -fill => 'red', -font => "courier");

}
                                    if ($_ eq 'G' or $_ eq
'g') {
                                                my  $iw  =
$canvas->createText(($h*10)+200,(($poo*10)+10),  -text  =>
"G", -fill => 'green', -font => "courier");

}
                                    if ($_ eq 'C' or $_ eq
'c') {
                                                my  $ie  =
$canvas->createText(($h*10)+200,($poo*10)+10,   -text   =>
"C", -fill => 'yellow', -font => "courier");

}
                                    if ($_ eq 'T' or $_ eq
't') {
                                                my  $ir  =
$canvas->createText(($h*10)+200,($poo*10)+10,   -text   =>
"T", -fill => 'blue', -font => "courier");

}
                                   if ($_ eq '-') {
                                                my  $it  =
$canvas->createText(($h*10)+200,($poo*10)+10, -text => "-
", -font => "courier");
                                                }
                                      }


                }
close(FH);
```

```
my $playbutton = $mw->Button(-text => 'Play',

                             -command => sub {

#########################
#When play button is pressed, create a new output file
and write
#Perl program in it that will create a MIDI file
#########################
open (FH, "$file2") or die "could not open file\n";
open (OUTPUT, ">$file2.out") or die "Cannot create
file\n";


foreach (<FH>){

$seq = $_;

s/a/A/g;
s/t/T/g;
s/g/G/g;
s/c/C/g;

unless ($seq = /^#/){

@array = split //;

$num = @array;

            for ($i = 0; $i < $num; $i++)
          {
          if ($array[$i] eq 'G' && $array[$i+1] eq 'T')
            {
            #push splice noise into sounds array;
            $sounds[$i]   =   "['note_on',   0,   1,   50,
96],\n['note_off', 200, 0, 1, 0],\n";
            }
          unless ($array[$i] eq 'G' && $array[$i+1] eq
'T')
            {
            #push rest into sounds array
            $sounds[$i]   =   "['note_on',   0,   1,   1,
1],\n['note_off', 200, 0, 1, 0],\n";
            }
          }
           #interpolate sounds array into string and push
into score array
          $sounds_string = "@sounds";
          #print OUT $sounds_string;
          push (@splice_score, $sounds_string);
```

```
#####################################################
#
# This section creates the array of note in DNAudio1
#
#####################################################
#    s/A/['note_on', 0, 1, 36, 96],\n['note_off', 200, 0,
1, 0],\n/g;
#    s/T/['note_on', 0, 1, 32, 96],\n['note_off', 200, 0,
1, 0],\n/g;
#    s/G/['note_on', 0, 1, 31, 96],\n['note_off', 200, 0,
1, 0],\n/g;
#    s/C/['note_on', 0, 1, 37, 96],\n['note_off', 200, 0,
1, 0],\n/g;
#     s/-/['note_on', 0, 1, 1, 1],\n['note_off', 200, 0,
1, 0],\n/g;
#    push (@score, $_);
#####################################################
    }
}
 close(FH);

#####################################################
#
#This section creates the array of notes in DNAudio2
#
#####################################################
#@sequence1 = split //, $score[0];
#@sequence2 = split //, $score[1];
#
#$leng= @sequence1;
#for (0...$leng){
#if   ($sequence1[$_]   eq   $sequence2[$_]){$s++;   push
(@scorep, "['note_on', 0, 1, 45, 96],\n['note_off', 50,
0, 1, 0],\n");}
#if ($sequence1[$_] eq "-" or $sequence2[$_] eq "-"){push
(@scorep, "['note_on', 0, 1, 20, 96],\n['note_off', 50,
0, 1, 0],\n");}
#else   {push   (@scorep,   "['note_on',   0,   1,   25,
96],\n['note_off', 50, 0, 1, 0],\n");}
#$score[0] = "@scorep";
#####################################################


$tracknumber = @score;
$splicetrackno = @splice_score;

#################################
#This is the child Perl file
#################################

print OUTPUT "use MIDI::Simple;\n";
```

```
print OUTPUT "use MIDI::Track;\n";
print OUTPUT "use MIDI::Opus;\n";
print OUTPUT "use MIDI;\n";
for ($j = 0; $j < $tracknumber; $j++){
print OUTPUT '$track';
print OUTPUT "$j = MIDI::Track->new;\n";
print OUTPUT '$track';
print OUTPUT "$j->events(\n";
print OUTPUT "['patch_change', 0, 1, 25],\n";
print OUTPUT "$score[$j]);\n";
}
for ($k = 0; $k < $tracknumber; $k++){
print OUTPUT '$track';
print OUTPUT "$j = MIDI::Track->new;\n";
print OUTPUT '$track';
print OUTPUT "$j->events(\n";
print OUTPUT "['patch_change', 0, 1, 25],\n";
print OUTPUT "$splice_score[$k]);\n";
}
print OUTPUT '$opus';
print OUTPUT "= MIDI::Opus->new(\n";
print  OUTPUT  "  {    'format'  =>  0,    'ticks'  =>  240,
'tracks' => [ ";
for ($j = 0; $j < ($tracknumber+$splicetrackno); $j++){
print OUTPUT '$track';
print OUTPUT "$j, ";
}
print OUTPUT "] });\n";
print OUTPUT '$opus';
print OUTPUT "->write_to_file('$file2.out.mid');\n";

close (OUTPUT);
##########################################################
####
#
#  Here  the  midi  file  is  created  by  running  the  perl
program written above
#
##########################################################
####


$status = system("perl $file2.out");


#########################################################

#open Winamp to play file that is given as an argument
#a "piped open" call is used to allow winamp to run as a
parallel process
```

```
#########################################################

open PLAYER, "|studio $file2.out.mid" or die "cannot pipe
to winamp: $!";
                                                    });
$playbutton->pack(-pady => 5, -padx => 5);

my $button2 = $mw->Button(-text => 'Exit', -command =>
sub {exit});
   $button2->pack(-pady => 5, -padx => 5);


}; #end display

######################
# SAVEMIDI
# copy file and rename it.
######################
sub savemidi {

$saveas = system("copy $file2.out.mid $savefile");
die "could not save file: $?" unless $saveas == 0;

};

######################
# INPUT
# Help on input
######################
sub input {
open INPUT, "input.txt" or die "could not open help file:
$!";
$inp = $mw->Toplevel();
$inp->title("DNAudio Help");
my $inphelp = $inp->Scrolled(qw/Text -setgrid true -width
40 -height 10
                  -scrollbars e -wrap word -spacing3 10/,
-font => 'Arial');
$inphelp->insert('end', <INPUT>);
#$inphelp->insert('end', "this will be really useful");
$inphelp->pack(qw/-expand yes -fill both/);

};

######################
# OUTPUT
# Help on output
######################
sub output {
open OUTPUT, "output.txt" or die "could not open help
file: $!";
```

```
$inp = $mw->Toplevel();
$inp->title("DNAudio Help");
my $inphelp = $inp->Scrolled(qw/Text -setgrid true -width
40 -height 10
                  -scrollbars e -wrap word -spacing3 10/,
-font => 'Arial');
$inphelp->insert('end', <OUTPUT>);
#$inphelp->insert('end', "this will be really useful");
$inphelp->pack(qw/-expand yes -fill both/);

};

######################
# DISPLAY_HELP
# Help on what is displayed
######################
sub display_help {
open DISPLAYHELP, "displayhelp.txt" or die "could not
open help file $!";
$inp = $mw->Toplevel();
$inp->title("DNAudio Help");
my $inphelp = $inp->Scrolled(qw/Text -setgrid true -width
40 -height 10
                  -scrollbars e -wrap word -spacing3 10/,
-font => 'Arial');
$inphelp->insert('end', <DISPLAYHELP>);
#$inphelp->insert('end', "this will be really useful");
$inphelp->pack(qw/-expand yes -fill both/);

};

######################
# PLAY
# Help on playing files
######################
sub play {
open PLAY, "play.txt" or die "could not open help file
$!";
$inp = $mw->Toplevel();
$inp->title("DNAudio Help");
my $inphelp = $inp->Scrolled(qw/Text -setgrid true -width
40 -height 10
                  -scrollbars e -wrap word -spacing3 10/,
-font => 'Arial');
$inphelp->insert('end', <PLAY>);
$inphelp->pack(qw/-expand yes -fill both/);

};

######################
# OTHER
```

```
# Additional help menu item if user evaluaion shows that
it is needed
######################
sub other {

$inp = $mw->Toplevel();
$inp->title("DNAudio Help");
my $inphelp = $inp->Scrolled(qw/Text -setgrid true -width
40 -height 10
                  -scrollbars e -wrap word -spacing3 10/,
-font => 'Arial');
$inphelp->insert('end', "this will be really useful");
$inphelp->pack(qw/-expand yes -fill both/);

};

######################
# HELPWIDGET
# help index with clickable text
######################
sub helpwidget {

$tl = $mw->Toplevel();
$tl->title("DNAudio Help");
my $helptext = $tl->Scrolled(qw/Text -setgrid true -width
30 -height 14
                  -scrollbars e -wrap word/, -font =>
'Arial');
    $helptext->pack(qw/-expand yes -fill both/);

    # Set up display styles

    my(@bold, @normal, $tag);
    if ($tl->depth > 1) {
     @bold     = (-background => '#43ce80', qw/-relief
raised -borderwidth 1/);
     @normal = (-background => undef, qw/-relief flat/);
    } else {
     @bold   = (qw/-foreground white -background black/);
     @normal  = (-foreground => undef, -background =>
undef);
    }

    $helptext->insert('0.0',    "DNAudio    Help    Files
Index\n\n");
    $helptext->insert('end','1. The Input File', 'd1');
    $helptext->insert('end', "\n\n");
    $helptext->insert('end', '2. The Output File', 'd2');
    $helptext->insert('end', "\n\n");
    $helptext->insert('end', '3. The display', 'd3');
    $helptext->insert('end', "\n\n");
```

```
    $helptext->insert('end',   '4.   Playing   the   Sound
encoded DNA', 'd4');
    $helptext->insert('end', "\n\n");
    $helptext->insert('end',   '5.   Available   for   other
help', 'd5');
    $helptext->insert('end', "\n\n");
    $helptext->insert('end',   '6.   Available   for   other
help', 'd6');

    foreach $tag (qw(d1 d2 d3 d4 d5 d6)) {
     $helptext->tag('bind', $tag, '<Any-Enter>' =>
            sub {shift->tag('configure', $tag, @bold)}
        );
     $helptext->tag('bind', $tag, '<Any-Leave>' =>
            sub {shift->tag('configure', $tag, @normal)}
        );
    }
    $helptext->tag(qw/bind     d1     <1>/     =>     sub
{&input('input')});
    $helptext->tag(qw/bind     d2     <1>/     =>     sub
{&output('output')});
    $helptext->tag(qw/bind     d3     <1>/     =>     sub
{&display_help('display_help')});
    $helptext->tag(qw/bind     d4     <1>/     =>     sub
{&play('play')});
    $helptext->tag(qw/bind     d5     <1>/     =>     sub
{&other('other')});
    $helptext->tag(qw/bind     d6     <1>/     =>     sub
{&other('other')});

    $helptext->mark(qw/set insert 0.0/);

$tl->Button(-text => "Close",
            -command => sub { $tl->withdraw })->pack;

}; #end help widget
```

Appendix 3: Genbank Accession Numbers

D32050.1
NM_064043.2
AF371466.1
AF371465.1

References

1. http://www.ncbi.nlm.nih.gov/Genbank/genbankstats.html
2. Thompson,J.D., Gibson,T.J., Plewniak,F., Jeanmougin,F. and Higgins,D.G. (1997) The ClustalX windows interface: flexible strategies for multiple sequence alignment aided by quality analysis tools. Nucleic Acids Research, 24:4876-4882.
3. Gotoh, O. (1996). Significant improvement in accuracy of multiple protein sequence alignments by iterative refinement as assessed by reference to structural alignments. J. Mol. Biol. 264: 823-838.
4. Morgenstern, B., K. Frech, A. Dress, and T. Werner. (1996). Multiple DNA and protein sequence alignment based on segment-to-segment comparison. Proc. Natl. Acad. Sci. USA .14: 290-294.
5. Thompson, J. D., F. Plewniak, and O. Poch. (1999). A comprehensive comparison of multiple sequence alignment programs. Nucleic Acids Research, 7(13): 2682-2690
6. Lesk, A.M. (2002) Introduction to Bioinformatics. Oxford University Press.
7. Kramer, G., Walker, B., Bonebright, T, Cook, P., Flowers, J., Miner, N., Neuhoff, J., Bargar, R., Barrass, S., Berger, J., Evreinov, G., Fitch, W.T., Gröhn, M., Handel, S., Kaper, H., Levkowitz, H., Lodha, S., Shinn-Cunningham, B., Simoni, M., Tipei, S. (1997). Sonification Report: Status of the Field and Research Agenda. Prepared for the National Science Foundation by members of the International Community for Auditory Display.
8. Hartmann, W. M. (1997). Sounds, signals, and sensation: Modern acoustics and signal processing. New York. Springer Verlag
9. http://www.casi.net/D.Quinn/bioaudio.html
10. Ohno,S. and Ohno,M. (1986) The all persuasive principle of repetitious recurrences governs not only coding sequence construction but also human endeavor in musical composition. Immunogenetics, 24, 71-78.
11. Hayashi,K. and Munakata,N. (1984) Basically musical. Nature, 310, 96.
12. King, R. and Angus, C. PM - Protein music. Computer applications in the Biosciences 12, 251-252. 1996
13. http://www.toshima.ne.jp/~edogiku/GSAwAD.html
14. http://search.cpan.org/author/NI-S/Bundle-Tk-1.00/
15. http://search.cpan.org/author/SBURKE/MIDI-Perl-0.8/lib/MIDI.pm
16. http://www.tigr.org/~jeisen/index2.html
17. Lidie, S., and Walsh, N. (2002) Mastering Perl/Tk. O'Reilly and Associates.
18. http://www.activestate.com/
19. http://www.winamp.com/