A-Level Coursework

# Analysis

## Inroduction

Learning a musical instrument takes a lot time. This time is generally divided up into two parts – Time spent with a teacher, and time to practice alone. A good balance of these is key to learning quickly and effectively. At our school, when learning a musical instrument, the student gets one half hour lesson a week one-on-one or two-on-one time with a music teacher who will then set pieces of music to practice for the next lesson. With this system, generally what happens is the student is intruded to a new piece of music in the music lesson which will then be practiced alone over the week until the next lesson, where the teacher will see how well they are coming along, and either introduce a new piece of music to the student if they had been able to play it well, or, spend the lesson listening to the student playing and giving feedback on how to improve. Unfortunately, due to the short lessons and the long wait between them, the student may be sent away to practice a piece of music over the week that they have not been able to understand in the lesson. This means that during the week, the student is faced with mastering a piece of music that they may never played all the way through and so do not how should sound or what they may be doing wrong and so by the next lesson, the student has not been able to make much progress learning the piece of music.

## Identification of End Users

The end users of the solution will be the students who are learning an instrument as well as the music teachers who are teaching the student.

After having a discussion with one of the music teachers, I have learnt that often when a student is practicing a piece of music alone, they may be learning the music incorrectly by repeatedly practicing something one way because it’s the way they think it should sound even if it is incorrect. Often students may just need a small bit of guidance to tell them that they are playing it incorrectly to start improving.

## User needs

* A program that will be able to show the user what they should be playing and listen to make sure that are playing the tune correctly
* Should be expandable to allow for a range of instruments to be used with it
* Should have a range of exercises that can be written and then imported at a later date

## Acceptable Limitations

* Due to time constraints and me not owning many instruments, it will only be tested with a single instrument, however will be written so the program can be easily expanded later.
* Teachers may not be able to add exercises that they have written themselves.

# Documented Design

## Overall System Design

The proposed solution will consist of 2 main features:

* The practice section - The student is shown the sheet music for the piece along with a several learning aids to allow them to practice learning and getting to grips with the piece of music
* The testing section – The student plays along to a piece of music and the program will listen to the audio and mark the student biased on how well they have played the song using several different metrics.

To meet the objectives well, the program will require:

1. A database of songs and instruments – The program should use a small database to store the information about the MIDI exercises that the student will be able to select. This will include: a link to the file on the disk; the difficulty of the exercise and the instrument used for the piece along with the instrument data itself. This will allow the student to select which song that they would like to practice for their instrument at the difficulty that they want to play. The database will need to store all the information that will be instrument specific, Eg: the range of notes it can play; the key the instrument is in; if the instrument transposes the music and the fingering for each of the notes.
2. The ability to read and interpret MIDI files – The program should use MIDI Files to store each of the exercises for later use. This means the program will need to be able to read in the MIDI files and be able to store the currently loaded exercise in memory in a way that will allow the other parts of the program to access the relevant information quickly and simply.
3. The ability to generate and display sheet music from the MIDI file -The program must be able to take the data from the MIDI file, generate and then display the corresponding sheet music for the student to read and play along to.
4. The ability to display several learning aids – The program should be able to display several learning aids to help the student follow along with the song. These could include: a metronome, a curser to show what note in the song they should be on and the fingering required for the student’s chosen instrument
5. The ability to listen to the audio being played and analyse the frequencies – The program must be able to listen to the audio that the student is playing and analyse it to work out what notes were being played at what particular time and save this data for further analysis
6. The ability to score how well the student has played – The program should be able to take the analysed audio and be able to grade how well the student played the piece by looking at various metrics. Eg: Was the right notes played? Did the note start at the right time? Did the note finish at the right point?
7. New exercised to be added – The program should allow the user to add new, exercises to practice. This will allow the user to download exercises that have been written after the program has been installed and add them to the library of exercises that they can practice.
8. A database to store instrument types – The program should have a database to store various information about each instrument Eg, the notes it can play, if the music is transpose or not and fingering diagrams for each note.

A possible extension would be a database for the students to track their scores over time, however due to time constraints, it is unlikely that this will be implemented in this version of the program.

## Componant explanation

### Main Componant 1 – Practice

* The student selects a piece of music to play from a list of catalogued pieces of varying difficulties
* The student is shown the sheet music and a selectable amount of learning aids. These may include the fingering for each note, a metronome to keep time, a cursor to show at what point in the song they should be on, ect. Having a selectable amount of help, will allow students who are just starting off to get used to playing the instrument. These can then be removed to allow the student to push themselves.
* The student can then play the music as many times as they please until they feel that they have mastered the piece

### Main Componant 2 – Testing

* The student is again allowed to select the music they wish to play from the list of pieces
* They select how many learning aids they wish to use to help them and then they play along
* The program uses the microphone to record the audio from the instrument. This allows the program to process the sound from the instrument to work out what note was being played, what time the note started and what time it ended. This can then be compared to the note that should have been played as stated by the music that the user selected to play. If the right note was played, at the right time until the right time, then the user would be given a higher than if the note was played wrong, or at the wrong time.
* At the end of the song, the user is shown the score they reciveced for each metric and shown which notes in the song that needs improving.
* The user would then be able to store in a simple database what score they got for each metric, the song they played and what helping aids they used. This database can be accessed later to see how they have progressed.

### Componant 1 – Database of Exercises and Instruments

This component will be responsible for storing the exercises that are available to the student to play and practice with and the instruments that they are written for. These will be specifically written MIDI files that will follow the MIDI Specification, but also meet a more specific set of criteria to be compatible with my program. Eg, all with only consist of one track chunk and only 1 instrument. This will make it easier for the program to deal with and allow the program to more easily process the file and display it to the user.

The database will also need to store various pieces of information about the instruments that the students can select to play from. These will need to

As an extension, files may be able to be created by a music teacher using existing MIDI software to generate these files which can then be transferred for the student to get a more tailored learning experience. However, for the time being, the program should be able to import midi files that I can create specifically.

@@ DATABASE DESIGN

### Componant 2 – Read and interpret MIDI Files

For the program to know what exercise the student is playing and what notes that are required to be displayed, the program will need to input the data as a MIDI file and then process it to store all of the notes and how long the notes will be played for.

To store the loaded MIDI file into memory, I will use a class. This class with be able to read in all of the MIDI data and convert it to a form that will make it simple and easy for other aspects of my program to access. The class will contain all of the methods that would be required to input and parse a MIDI file and provide validation on the MIDI file as it is parsing it to make sure that no erroneous data has been added.

|  |
| --- |
| midiFileClass |
|  |
|  |
|  |
|  |
|  |

### Conponant 3 – Generate Sheet music

### Componant 4 – Learning aids

### Conponant 5 – Anayse Frequencies

### Componant 6 – Score playing

### Componant 7 – Add new exercises

## IPSO chart

|  |  |  |  |
| --- | --- | --- | --- |
| Input | Process | Storage | Output |
| Midi File to display selected | Read MIDI File in | Store the MIDI file in memory in a format my program can understand | Show the user the sheet music for the midi file |
| Audio from the michrophone |  | Store the input audio in a wav file not in ram |  |
| Wav file | Analyse what note was being played at each time interval | Store what notes were being played at each time in a list in RAM |  |
| List of what notes were being played at what time | Compare the list of notes to the midi file to see if the right notes were being played at the right time |  | Show the user a score to show how well they did playing the piece |

## Modular Design

* **First Form** (Main Menu)
  + Select instrument
  + Select music
  + View sheet music
    - **Second Form** (Display sheet music)
      * Next page
      * Previous page
      * Main menu
  + Select practice
    - **Third Form** (Practice)
      * Previous page
      * Previous line
      * Play/Pause
      * Next line
      * Next page
      * Main menu
  + Select Test
    - **Fourth Form** (Test form)
      * Start
      * Pause
      * Stop
      * Retry
      * Main Menu
      * Mark
        + **Fifth Form** (Result Form)

Retry

Main Menu

Quit Program

## Data Requirements

The program will need to store quite a bit of data in order to function.

This only convers the main, bulky, bits of data that will need to be stored processed. Is not an exhaustive list of all variables.

|  |  |  |  |
| --- | --- | --- | --- |
| What is being stored | Why it is being stored | Where is it being stored | How long will it be stored |
| Selection of songs for the student to play | The songs will need to be stored for the user to play back when they need them | The songs will be stored in a series of folders which will be linked to through a small database | The songs will need to be stored for as long as the user wants, across sessions. |
| Audio input from the microphone | The audio from the microphone needs to be stored so that the program can run through and process it to find what notes were being played at what time to give the student a score | The recorded audio will be stored in a wav file in the main program storage to make sure a long song does not use up excessive amounts of ram | The audio will only need to be stored from the start of the recording to the end of the processing of the audio |
| The song that is currently being played | The will need to be stored to allow the program to generate the images of the sheet music what will be displayed to the user and also be compared to the recorded audio to find out if the student was playing the right notes at the right time | The current song will be stored in a custom class that will allow the program to access the data it needs quickly. It will be stored in the RAM | The current song will need to be stored from when it is first loaded in by the user until the program is closed or another song is selected and loaded into memory |
| The notes that were played in the recorded audio | After the audio has been processed to find out what notes were being played when, the notes that were being played need to be stored to be compared against what the midi files says should have been being played | This will be stored in a list in the RAM for quick access and easy processing. | The notes that were being played will only need to be stored from when the audio has been processed until the score has been given to the user |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

### Data Validation

The program will need to validate the MIDI file as it is being inputted into the program to make sure it is the right format and only contains 1 instrument track to make sure that a error is not thrown later down the line when the file is being processed. If the file is the correct type and is not corrupt, then the program will load the file into memory and then allow the program to continue.

The selection of midi files will be stored inside folders using a small database to allow the user to find and load the specific tasks of whatever difficulty they wish that are applicable to what ever instrument they wish to play. When the program starts, it should run through the database and make sure that the midi files referenced in the database can be found in the file system. If, for whatever reason, the file is missing, it should either not display that record to the user or delete the record entirely.

For anything the else that the user will be selecting, I will be used controls where the user does not get to type in their own values. Eg, drop down lists and sliders. This limits the amount of validation that I will have to perform on the data and make the program more robust.

## Preliminary Form Sketeckes

### Form 1 – Main menu

On Load

lblInstument

btnTest

btnDisplayMusic

btnQuit

btnPractice

btnOpenFile

Select a song:

Instrument: \_\_\_\_\_\_\_

Test

Quit

Practice

Look at music

Open File

###### Process

* Check all files referenced in the database are there and hide ones that are not found
* Query database to get list of different instruments available
* Display instrument list in Combo Box
* Query Databse to select all exercises and display them in the datagrid view

###### PsuedoCode

*numOfExercises ← getNumberOfExercises*

*FOR i = 0 TO numOfExercises*

*currentFilePath ← queryDatabase("SELECT Path FROM Exercises WHERE id=i")*

*IF checkFileExists(currentFilePath) = true THEN*

*databaseNonQuery("UPDATE Exercises SET FileExists = true WHERE id=i")*

*ELSE*

*databaseNonQuery("UPDATE Exercises SET FileExists = false WHERE id=i")*

*END IF*

*NEXT i*

*comboBoxOptions ← queryDatabase("SELECT DISTINCT Instrument FROM Exercises WHERE FileExists=true")*

*dataGridViewData ← queryDatabase("SELECT \* FROM Exercises WHERE FileExists=true")*

comboBoxInstrumentChanged

###### Process

* Query Database to select all exercises where the instrument used is the instrument selected
* Assign the result to the data grid view

###### PseudoCode

*dataGridViewData ← queryDatabase(“SELECT \* FROM Exercises WHERE FileExists=true AND Instrument=comboBoxSelectedInstrument”)*

btnDisplayMusic

###### Process

* If file has been selected,
* Open DisplayMusicForm and pass through midiFileObject
* Hide main Menu

btnPractice

###### Process

* If file has been selected,
* Open Practice form and pass though midiFileObject
* Hide main Menu

btnTest

###### Process

* If file has been selected,
* Open Test form and pass though midiFileObject
* Hide main Menu

ImportSong

###### Process

* Shows the open file dialogue
* Allows the user to select a Midi File to open
* Converts the MIDI file to a byte stream
* Reads the head chunk to make sure the file is compatible, displaying an error if it is not
* Reads the rest of the file parsing it into the class that with store the file, catching any errors that occur
* If it’s successful, then the instument that the file is written for will be displayed on “lblInstument”

###### Pseudo Code

*Show OpenFileDialogue*

*byteStream = selectedFile*

*FOR i = 0 to 13 LOOP*

*headChunk = headChunk + readByteFromStream(i)*

*NEXT i*

*IF verifyFormat(headChunk) = true THEN*

*midiObjectParse(byteStream)*

*ELSE*

*Message “Format not compatible”*

*ENDIF*

btnQuit

###### Process

* End the program

### FORM 2 – Look at sheet music