€

Candidate Number: 7629

Jonathan Broster

A windows app that gamifies the process of learning piano chords by listening to the player

H446

A-Level Computer Science

A-Level Computer Science

Practical Programming Project

Contents

[Analysis 2](#_Toc147832692)

[Summary 2](#_Toc147832693)

[Further Problem Identification 2](#_Toc147832694)

[Computation 2](#_Toc147832695)

[Decomposition 3](#_Toc147832696)

[Divide and Conquer 3](#_Toc147832697)

[Abstraction 3](#_Toc147832698)

[Data Mining 4](#_Toc147832699)

[Threading 4](#_Toc147832700)

[Conclusion 4](#_Toc147832701)

[Stakeholders 4](#_Toc147832702)

[Interviews 4](#_Toc147832703)

[Research 6](#_Toc147832704)

[Conclusion 8](#_Toc147832705)

[Essential features 8](#_Toc147832706)

[Display Chord 8](#_Toc147832707)

[Listen to Notes 8](#_Toc147832708)

[Gather Data 8](#_Toc147832709)

[Learning Algorithm 8](#_Toc147832710)

[Sheet Music 8](#_Toc147832711)

[Limitations 8](#_Toc147832712)

[Time 8](#_Toc147832713)

[Melody/Tempo 8](#_Toc147832714)

[Optional Features 9](#_Toc147832715)

[Song mode 9](#_Toc147832716)

[Piano Graphic 9](#_Toc147832717)

[Requirements 9](#_Toc147832718)

[Testing and Development 9](#_Toc147832719)

[Success Criteria 9](#_Toc147832720)

[Design 13](#_Toc147832721)

[Decomposition 13](#_Toc147832722)

[Structure of the Solution 14](#_Toc147832723)

[Listener 14](#_Toc147832724)

[Accepting microphone input 14](#_Toc147832725)

[Translating the sine wave 14](#_Toc147832726)

[Picking out loudest frequencies 15](#_Toc147832727)

[Converting the frequencies to note names. 15](#_Toc147832728)

[UI 16](#_Toc147832729)

[Sections of the UI 17](#_Toc147832730)

[Design of the UI 17](#_Toc147832731)

[Revision engine 17](#_Toc147832732)

[Classes 18](#_Toc147832733)

[Listener 18](#_Toc147832734)

[Development and testing 19](#_Toc147832735)

[Evaluation 19](#_Toc147832736)

# Analysis

## Summary

I would like to create a windows app that gives a piano chord then listens for the chord to be played, then automatically gives a new chord.

## Further Problem Identification

Currently there is not a good app to help you practice playing chords on the piano. It is vital to practice playing chords to become a better player, but it can become boring and repetitive. Therefore, I want to create an app to gamify the experience of learning new chords by timing how long it takes you to play the chord. This app could also keep track of what chords you are competent in, and show them less, and what chords you are not competent in playing and show them more. This competency rating can be calculated from the time that it takes you to play the chord. Through this, it can help the user to learn the chords through a fun game-like approach rather than boring repetition and practice. This will increase user satisfaction with piano learning, making it a more satisfying and interesting experience.

## Computation

The solution is applicable for computational methods for many reasons. Firstly, the process of listening for a sound and comparing it to a known sound or note can be done quite easily by inputting a section of sound and doing a Fourier transform to convert the wave to a list of frequencies present. I can then discard all the frequencies whose volumes are below a certain threshold, and then convert the remaining frequencies to notes. The computer can automatically display a new chord if the correct notes are detected, whilst keeping track of all the previous chords played and how quickly, to train the user to play the chords much quicker.

### Decomposition

The project can be split into three main parts which should be able to work independently of each other:

* Microphone/Listener/Note Identifier
* UI
* Game – question storage and tracker

By splitting the project in this way, I can ensure that the project can be adapted to meet the needs of any other potential clients other than the stakeholders, so that the code is versatile, modular and works on many devices. I should also be able to employ abstraction in these three areas, so that when developing the UI, I do not need to code the game directly, I only need to interface with it. Furthermore, it will make the project easier to debug because I will be able to more easily identify in which section of the project the bug is in, rather than having to debug the entire code.

The solution involves an algorithm which has some steps:

1. Listen for a sound.
2. Convert the sound from a sine wave to a frequency chart by Fourier transform.
3. Scan the frequencies found to compare against expected frequencies of notes.

This is how the computer will detect chords that the user will play. To single out any notes from background noise, I will only accept the frequencies whose amplitudes are above a certain threshold.

### Divide and Conquer

Solving the above decomposed problems together seems computationally intensive. To be able to write my solution to the problem efficiently and easily, I will need to conquer each of the decomposed problems separately. I will even divide these components into smaller algorithms and subprograms that seem more manageable on their own. For example, I will first handle the background tasks such as microphone to notes before I do the game and then the UI.

Each component in the abstraction section can be programmed separately so that they fit together modularly. This will make it easier to expand on the app in the future, and to build each section of the app.

### Abstraction

Each part of the solution will be abstracted from the others. At the top, there will be the UI, and what the user sees. Then, underneath that there will be the game, which will load new chords and keep track of the user’s competency of the chords. Underneath the game, there will be two components: the file system and loading, because it would be a good idea to store the chords in secondary storage so that the user can save their progress. Also, there will be the note identifier, which takes in frequencies and compares it to notes then outputs any notes that it hears. There will be one more component underneath that which listens and converts the sine wave to frequencies.

My reasons for splitting it this way is that these “modules” are separate to each other and although they need each other to run, they do not need to know what the other modules do. I can program them completely separately, so that I can modify one of the modules and it won’t affect how the other modules run. Furthermore, if I wanted to add more functionality in the future, I would just add more modules the current tree, and I would not need to change much of the original modules to do that.

### Data Mining

The project will implement a simplified version of data mining where I will collect data such as time to find each chord, how many wrong notes were played before the chord was detected, etc. I can then use this data to show the user chords that they find more difficult to play more often, so that they will learn faster.

### Threading

To solve my problem, I will need to utilise multiple threads, as the computer will need to be listening, counting, and checking the previous notes at the same time. My threads will need to pass data to one another so I will implement a lock on some global variables to allow this. One example of where this will be useful is the thread that is listening passes the frequencies to the game, to check that the notes are correct.

### Conclusion

Because of all the above-mentioned computational methods, the solution is very clearly solvable by computational methods. In fact, the solution can only be solved effectively by computational methods because a human would not be able to identify the chords or provide new ones with enough accuracy.

## Stakeholders

## Interviews

### Key questions for James & Sarah:

1. Are you satisfied with how you currently learn to play piano?
2. Have you ever tried an app to help you learn, and were you satisfied with how the app helped?
3. Do you often play by chord patterns or do you play by reading the music in its entirety.
4. Do you feel like an app to help you practice a specific area of piano would be better than a general-purpose app?
5. Would you prefer more of a game-y app or a revision app?
6. Is there anything that you would love to see in an app like this?

Questions 1 and 2 establish whether they are happy with how they learn. This is important because it depends on this how they use the app, and whether it completely re-shapes their learning or just is a fun game.

Questions 3 and 4 enquires about a need for a specific app that is for more advanced players, rather than a “learn piano from scratch” kind of app.

Question 5 helps me to understand how game-y the app should be. This is important because some apps go to far, and some don’t go far enough. For example, you don’t want to be running around a 3D world whilst playing your chords, it’s too far, but also, you don’t want to be staring at a grey screen telling you chords to play with no reward or game at all.

### Key questions for Mr Johnson:

1. Are you satisfied with how your tutees currently learn to play piano?
2. Have you ever tried an app to help you teach, and were you satisfied with how the app helped?
3. Do you often teach to play by chord patterns or do you prefer playing by reading the music in its entirety.
4. Do you feel like an app to help teach a specific area of piano play would be better than a general-purpose app?
5. Would you prefer your students to learn from more of a game-y app or a revision app?
6. Is there anything that you would love to see in an app like this?

Questions 1 and 2 establish whether they are happy with the resources available for teaching. This is important because it depends on this how they use the app to help their students, and whether it completely re-shapes their teaching or just is a fun game.

Questions 3 and 4 enquires about a need for a specific app that is for more advanced players, rather than a “learn piano from scratch” kind of app.

Question 5 again helps me to understand how game-y the app should be.

### James

1. No because at the moment I do not have time to practice as much as I would like, however I feel like when I do have the time, I can improve quickly in pieces I’m learning. I do find that my lessons with my teacher every Friday are very valuable to me though.
2. No I haven’t because my teacher is very good and trained me in classical piano and therefore, I do not feel that an app would improve my ability.
3. I find hat reading music in its entirety is the only way to play classical piano, chord patterns are more useful in Jazz/blues music.
4. I would prefer a general app that can be used to also practice specific areas such as scales, sight reading, understanding music etc.
5. Revision because I am more bothered about targeted focus and serious practice.
6. Sight reading trainer because being able to sight read well means learning the notes to new pieces becomes much easier although of course the finer details can take just as long to master.

### Sarah Smith

1. No, not really. I can’t play my favourite songs, and it’s taking too long to learn the chords. I use online lessons to try and improve but they’re quite boring.
2. I watch YouTube tutorials to learn piano because, being in college, I haven’t enough money for a private tutor or a subscription-based app.
3. I do a bit of both really. If the chords on the music look a bit hard, I’ll figure them out from the letters, and if I don’t know what the letters mean, I’ll usually work it out from the music or look it up online.
4. Definitely! I feel like the general-purpose apps don’t tailor to the needs of the play at all. I’m quite good at timing and melody so I don’t need to practice that as much.
5. I’m a huge fan of anything that makes learning more fun, so definitely a more game-y app.
6. I’d love to see colour-blind support and easy to recognise visual representations of the chord patterns, because I’m colour-blind, and I have dyslexia.

-----

These stakeholders an interested in a game-revision hybrid, that is fun to learn, but also makes sure you do a suitable amount of learning. They are not interested in any other game components that wouldn’t contribute to their learning.

Sarah pointed out that there needs to be accessibility features such as colour-blind settings, and visual arrangements of the chords for those who have dyslexia.

Ian said that he didn’t have much time to learn, so the sessions need to be quick and concise, with no ads or annoying pop-ups.

### Mr Johnson

1. Mostly. My students sometimes get a bit bored memorising theory, and they just want to be playing songs, but unfortunately you need to know the chords to play the songs.
2. I’ve looked at some, but they’re only good for complete beginners, “find the note” type apps, if you know what I mean.
3. I teach both, as they are both important skills to have. Students should be able to read the music but also it can be quicker to read the chords if the player is competent enough. It entirely depends on which they are better at.
4. I hate the general-purpose apps, because they act like they can replace a tutor, but they don’t teach half the stuff like chords, posture, etc.
5. I think they’d love the game-y app, but they do need to learn, so it needs to be fun but also educational.
6. I would love to see the ability for me or the student to select chords they want to practice more that session, so that I can set them tasks to practice certain chords.

----

Mr Johnson seemed to agree with the other stakeholders, that the app needs to be fun but educational. He also highlighted the need for there to be some control over the chords that are learnt in a session. Furthermore, he pointed out that the app should be a tool to help, not to completely overthrow teachers.

## Research

#### “Yousician”

A screenshot of a music website

Description automatically generated

* An app that helps people learn various instruments by showing the notes and listening to the user play them. Doesn’t focus on chords, but more like reading traditional sheet music.
* Things I like and could incorporate into my app:
  + “Yousician” plays the chord so that you can learn to hear what it should sound like. This is needed so that the user can learn to listen to the chords, and this helps ear training.
  + Gamify, score system. This keeps the user engaged with the app for longer so that they can learn more.

A screenshot of a computer

Description automatically generated

* + Piano at bottom of screen showing notes. This helps beginners to find the notes quicker and see the chord patterns.

A screenshot of a computer

Description automatically generated

* Things I don’t like:
  + Too many tutorial videos. The user should spend most of their time practicing than watching videos, and the learning points should be developed into the game.
  + Tailored much more towards beginners. I need my app to be catered towards many difficulties to suit many types of piano players.

A screenshot of a computer

Description automatically generated

(These notes are far too simple)

* + Too much background tune. This tune can be fun for the user but it’s harder to recognise the chords whilst learning and the melody should only be played when the song itself is in practice mode, rather than all the time.

#### “Simply Piano”

A screenshot of a video game

Description automatically generated

* A large, versatile piano app that helps the user to learn how to play popular songs. Overly broad range of things you can do on it, not great for more advanced learners.

A screenshot of a music app

Description automatically generated

Versatile Piano App



The notes here are too simple.

* Provides songs based on the courses that the user has completed.
* Shows sheet music, and helps you learn all aspects of music from the very beginning.
* Some things I could take from this include:
  + The sheet music to show which notes to play. This can teach the user to read the music as well as play the chords.
  + Videos/cartoon graphics showing which notes to play. These should be short and concise to keep engagement high.



* + The learning curve and complexity scale. This is a good example of how to tailor an app to multiple difficulties and react to how easy the user finds the chords.
  + The UI is neat and minimal – quite simple and no way to get confused. This is good for users with a range of technological experience.
* Some things I don’t like about Simply Piano include:
  + Only available for mobile devices, which tend not to have as good a microphone as a computer with a stand-alone microphone. Also, mobile devices often do lots of hardware manipulation to the microphone to try and single out the voice for calls. This is less good if you want to record not a voice.
  + You can’t skip around the music or slow down and speed up the tempo.
  + Expensive subscription.
  + No option to turn of backing track. The backing track was annoying especially when trying to play.

### Conclusion

Based on this research, I can conclude that I do not want to over-complicate the app, as that is the downfall of many of the larger piano-learning apps. They try to listen to whole songs and rhythm too, and most of the time it doesn’t work very well. There are a few apps which attempt to do something similar, but they don’t often do exactly what my stakeholders want and are often bloated and do lots of things not very well instead of one thing very well.

## Essential features

### Display Chord

The chord name (e.g. Cmin11) should be displayed boldly as the main subject in the screen. The chord name should update on each round of the game when the user moves on to the next chord. This can either be by getting it right or getting it wrong. This is essential because it is the core basis of the game, and it is how the user sees what they need to do.

### Listen to Notes

After a chord is displayed, the app needs to listen to the user playing notes to deduce if the user is playing the correct notes. This can be done by listening to tiny amounts of audio regularly through the microphone and doing a FFT to determine the frequencies present, and then checking the frequencies to make sure the notes are correct. This is essential because it is how the user interacts with the game and it makes it so that the user can play whilst the app listens.

### Gather Data

Whilst the user is playing, the app should gather data such as how long it took the user to get the answer correct and how many tries it took, or incorrect notes. This is so that the app can get an idea of which chords the user knows better than others and give these chords less. This is an essential feature because it helps the user to learn by gathering data about their performance.

### Learning Algorithm

The app should then use the data collected to give the chords to the user in a better order, giving some chords more frequently depending on the user’s apparent competence in the chord. This is justifiable because the clients said they need a way of tailoring the chords that are asked to the ability of the user. This is an essential feature because it helps the user to improve by feeding them chords that they find harder.

### Sheet Music

As per my stakeholder’s request, there should be an option to play the chords by sheet music as well as chord name. This is an essential feature because it will help the user to learnt to recognise the chords on music, as per the stakeholder’s request.

## Limitations

### Time

One of the biggest limitations is time. I have only a couple of months to design and create this app, so some features must be left out. However, in the section labelled “Optional Features”, there are features that could be implemented if I have time, features that would be nice to have but aren’t essential.

### Melody/Tempo

The app will not have capacity to recognise the tempo or rhythm of the user’s playing, just the notes that they play. This is because it is difficult to measure a melody because students play at a wide variety of speeds. Furthermore, chords don’t require rhythm or tempo to play when they are not in the song, so it is purely outside of the scope of the app, which doesn’t teach the user songs but the chords.

## Optional Features

### Song mode

The user could play along to a song that the app plays and displays the chords next to it. This could be a fun game for the user and the user could even learn the melody then not even need the chord. This is an optional feature because it is non-essential to the user learning chords and it can be done without an app. However, if a song mode were to be put in, it would upgrade the user experience.

### Piano Graphic

There could be a piano graphic at the bottom of the screen which highlights keys based on which ones the user should play kind of like a typing game. This is a feature that could be hard to implement, and as the stakeholders have requested it, it will be implemented if there is enough time.

### Dark Mode

This would be a nice to have feature because people prefer it to light mode as it saves their eyes so that they can look at it for a longer time. This is optional because the stakeholders wanted something like this, but again it could be quite time consuming to implement.

## Requirements

### Hardware and software

|  |  |
| --- | --- |
| **Requirement** | **Justification** |
| Windows OS | Because I am going to program it as a windows form application, which I have chosen because it is the most compatible OS. |
| Standard computer | The user needs a computer to use this software. This includes up to date hardware and a small amount of spare storage space. They will also need standard peripherals if they are not using a laptop, e.g. mouse, keyboard. |
| A microphone | The user will need a microphone so that the app can detect the notes that they play. The better the microphone, the more accurate the notes will be. |
| .exe file compatibility | I will package the file as an installed app that the user can install using an installer or a portable app that the user can just run. Either way, their computer will need to have support for exe files. |

### Testing and Development

Visual Studio C# Win Forms Application – a useful IDE to develop and test the code. “The Visual Studio IDE is a creative launching pad that you can use to edit, debug, and build code, and then publish an app. Over and above the standard editor and debugger that most IDEs provide, Visual Studio includes compilers, code completion tools, graphical designers, and many more features to enhance the software development process”[[1]](#footnote-1)

## Success Criteria

|  |  |  |  |
| --- | --- | --- | --- |
| Criteria Number | Description | Justification | Achieved? |
| 1 | Note Detection |  |  |
| 1.1 | The program listens and saves the recording in RAM | Needs to be able to listen to the user |  |
| 1.2 | The program converts the sine wave to frequencies | Needs to be able to interpret the sound |  |
| 1.3 | The frequencies are translated into notes | Needs to be able to figure out which notes are being played |  |
| 1.4 | The notes are saved, and duplicate notes are not | So that the program can read the notes and compare them to the notes it expects to hear |  |
| 1.5 | Background noise is not counted as notes | The program needs to be able to convert only the notes that are most prominent. |  |
| 2 | Difficulty/Revision – Choice of chords |  |  |
| 2.1 | The program stores lots of chords, and the chord’s notes in a large structure. | So that the chords are available quickly and an algorithm can figure out which chords to choose |  |
| 2.2 | The program chooses a chord based on some data collected | So that when the user starts getting it correct quickly, the chord plays less |  |
| 2.2.1 | Each chord has a “score” which is how well the user knows it |  |  |
| 2.2.2 | The score is re-calculated each round based on data |  |  |
| 2.3 | Chords are chosen by score, but never repeated directly (can be ABAB but not AABB) | So the user doesn’t get the same chord more than once in a row |  |
| 2.4 | The user can select a difficulty | So that advanced users don’t have to start on easy chords |  |
| 2.5 | The user can set favourite chords which have a much higher “score”, so they are played more | So that the piano player can select specific chords they want to practice, for example the tutor wants the student to practice the “Gmin” chord patterns |  |
| 3 | Game |  |  |
| 3.1 | The game gives the user a chord | So the user knows what to play |  |
| 3.2 | The game checks any notes currently being played against the chord notes | So the game knows if the notes are correct |  |
| 3.3 | If the notes are correct, the game gives the user a new chord and the correct screen is shown. The time is reset. | So the user knows if the notes are correct |  |
| 3.4 | There is a score that counts how many chords the user has correct, like a streak | This helps the game be more of a game and less a revision tool |  |
| 3.5 | The game shows the user the correct answer after they press help | This helps the user to learn the chord |  |
| 3.6 | The user can favourite a chord mid-game | So that the user can go back and reflect upon the chords |  |
| 3.7 | The user can press “next” to skip a chord | So that if the user doesn’t know they are not stuck in a loop |  |
| 3.8 | At the end of the time limit, the answer is “incorrect” | So that the game moves on if the user doesn’t know. |  |
| 4 | UI |  |  |
| 4.1 | The GUI is clean and simple and scalable. | So that people who might not know how to use computers competently can still use the app |  |
| 4.2 | The streak is displayed | This is purely for gamification |  |
| 4.3 | The current chord is displayed | .. |  |
| 4.4 | There is indication that the chord is favourited, difficult, etc. | So the user knows whether they have favourited it or not |  |
| 4.5 | The background changes to green with ticks when the answer is correct, and red with crosses when the answer is incorrect. | This is because one of my stakeholders is colour blind, so they won’t know the difference between read and green. However, for those non-colourblind it still needs to be visually appealing and easy to understand briefly |  |
| 4.6 | 4.5 is easy to understand for a colour-blind person | .. |  |
| 4.7 | Next button/Help button | For the user to select the next chord just in case they don’t know the answer |  |
| 4.8 | Some indication of the correct chord visually | So the user can more easily find the chord on the piano |  |
| 5 | Catches/robustness |  |  |
| 5.1 | No microphone – recommends to the user that they should plug in a mic but continues anyway.  This should also trigger if there is little or no sound | This is so that the user can practice their recall of the chords like flashcards if they are out and about. |  |
| 5.2 | Badly Tuned piano – does its best but doesn’t respond to notes that are too far away from the accurate frequency | This is so that there is less chance of an error |  |
| 5.3 | All inputs validated | So there is little chance of crashing if the user wants chord number -1 to be displayed. |  |
| 5.4 | Notes that are too loud or quite should be dealt with  Use of a decibel scale could help here. | If the notes are too quiet on average, I still need them to be picked up by the microphone, and if there is some background noise then some loud notes, I need to differentiate between them so that the background noise isn’t perceived as notes. |  |

# Design

## Decomposition

I am going to break the problem down into a series of smaller problems, which are individually suitable for computational solution. Below is a diagram showing the different modules that I am going to need to build.

At the top of the tree is the app, which splits into the “Main Game”, which contains all the background coordination and timing, and the “UI”. I have split it like this because the UI should not need to know anything about the game or background application and should work as on its own. Moving depth first, under the “Main Game”, there is a timer to help the game run and to track the user’s response time. There are background processes include the revision engine, which controls the progress and which chords are selected for the user, and the listener, which detects microphone noise. As per my stakeholder’s needs, the revision engine gives chords based on the chords that have previously been played, and the user’s confidence on them. Built into this is the favourite system, which one of my clients requested. Another main part of the background processes is the listener, which controls the microphone and input and converting that to notes. I have put this at the bottom of the tree because the game, UI, or anything else does not need to know how it works, only the output from it.

Under the UI section of the game, there is all the components that need to work together for the UI. The reason I have added the microphone into this is because I will need to plot microphone levels for debugging purposes. A scaled down version of this can be in the final app to help the user to debug their microphone.

## Structure of the Solution

I am going to use a more agile method and therefore I will regularly check in with my stakeholders to get their feedback on the solution. This is so that I can ensure that I am making the app that they want.

### Listener

This section of the solution will oversee the gathering and processing of data from the microphone. It will have 4 different parts:

#### Accepting microphone input

This algorithm will put any available microphone input into a buffer that can be operated over. The data will be in the domain of time and will look like a sine wave.

#### Translating the sine wave

I am going to use PCM and a FFT to complete this.

PCM (pulse-code modulation) is a way of digitally representing analogue signals. The amplitude of the signal is sampled at uniform intervals and quantized to the nearest value within a range of steps.[[2]](#footnote-2)

FFT (fast Fourier transform) computes the Discrete Fourier transform of a sequence. This process converts a signal from its original domain (e.g. a time-domain graph shows how a signal changes over time) to a representation in the frequency domain (how a signal is distributed through many frequencies).[[3]](#footnote-3)

A screenshot of a graph

Description automatically generated

A discrete Fourier analysis of a sum of cosine waves at 10, 20, 30, 40, and 50 Hz  
(<https://en.wikipedia.org/wiki/Fast_Fourier_transform>)

There are many different types of FFT. I am going to use the Cooley-Tukey Algorithm, and to do that I am going to use a library called Accord.Math which has a function that takes an input of the data as an array of complex numbers (it uses complex numbers so that it can represent the phase of the wave as well as the amplitude) and a direction, which will be forward in this case.

Because I am only interested in frequency, however, I can discard the second half of the values in the array. This should output an array of values where for each array index, is a different frequency increasing in step (calculated by the max frequency and the buffer size to fit the size of the structure).

#### Picking out loudest frequencies

This algorithm should iterate over the array and pick the most prominent frequencies. This could be any frequencies above a certain amplitude. The problem with having that though, is that any background noise might also be quite loud, and I need to distinguish between the background noise and the actual notes. There are two main ways to help pick out these frequencies:

The first way is to use a logarithmic system like a decibel scale so that the frequencies that are louder appear much louder than the rest. A second way would be to discard any frequencies that are too far away from actual notes. This will help to detect in-tune pianos more accurately.

#### Converting the frequencies to note names.

This is done by these equations:

This equation works out the MIDI number (fig. 1) by comparing it against the frequency of a known note, A, which has frequency 440Hz, and is MIDI number 69. We use log base 2 here because if you double the frequency of a note, you get the same note one octave higher (A5 would be 880Hz). This means that the scale for increase of pitch is logarithmic not linear. This is because our perception of pitch is also logarithmic.  
The MIDI Number should be rounded to a whole number here (Success Criteria 5.2)

To work out the actual note name from the MIDI number, we can just use a list of all the note names:

{ "A", "A#/Bb", "B", "C", "C#/Db", "D", "D#/Eb", "E", "F", "F#/Gb", "G", "G#/Ab" }

And go to the index worked out here:

We take 21 because the smallest MIDI number (A0) is is 21. Then modulo because there are 12 items in the list and I don’t care about the octave that is played.



Figure 2

https://newt.phys.unsw.edu.au/jw/notes.html

#### Listener Class

This class oversees the process of getting the notes from the sine wave at regular intervals. It contains some constants, including the rate, buffer size and device number. These constants make sure that all the data can be modified in the same way because the sound waves have the same properties. This class also has many methods:

|  |
| --- |
| Listener |
| + WaveIn waveIn;  - BufferedWaveProvider bwp;  - int RATE;  - int BUFFERSIZE;  + int DEVICENUMBER; |
| + Listener()  - waveIn\_DataAvailable()  + StartListening()  + string WhatNoteAmI()  + List<string> ProcessData()  + double[] CalculatePCMValues()  + double[] CalculateFFTValues()  + double[] DbScale() |

#### Listener

This is the constructor for the algorithm, and it crucially allows the setting of the wave properties and constants in the attributes section. It also contains default values for the properties that match what is expected for the wave functions to be, e.g. 44100kHz, device number 1. This method will also create a buffer that the sound will be temporarily stored in whilst it is scanned.

##### waveIn\_DataAvailable

This is an event handler for if there is any data available from the microphone. It has the simple job of adding the samples to the buffer so that another process can read it.

##### StartListening

This is a method that starts the recording of audio from the mic. More specifically, it should initialise the buffer and set the waveIn device to recording.

##### WhatNoteAmI

This method works out, from the frequency that it is given, the note that is played by using the method mentioned above (“Converting the frequencies to note names”).

Pseudocode:



##### Process Data

This method will regularly be called to process the most recent data in the microphone. It will call other methods to help it do this.

A diagram of a graph

Description automatically generated

##### Calculate PCM Values

The method has an input of a byte array containing the information in the buffer, and how many points of data there are. The method should then output a double array of values. These values are the integer numbers stored in every 2 bytes of the array as a percentage of the maximum 16-bit integer.

This array is then the output.



##### Calculate FFT Values

Similarly to PCM values, this method takes an input of the byte array and the number of samples of data. It also takes an input of the PCM array. This method will change the sine wave to frequencies by using the method mentioned above (“Translating the sine wave”).



##### DbScale

This function should take the FFT values and provide a new array of doubles that are modified by a logarithmic scale, like how a decibel scale works. This should make sure that the most prominent frequency’s amplitude stands out much higher than the background noise so it will be easier to distinguish and identify the notes. It might also be a good idea at this point to discount frequencies that are too far from actual notes to further help to identify the real notes. See above (“Picking out loudest frequencies”).



### UI

This section of the solution will oversee the UI. The UI should be coded completely separately and modularly.

#### Sections of the UI

##### Game

This part of the GUI contains buttons and displays that the user will need to press and view to control the game. This includes: the next button, which allows skipping a chord; the stop/restart button, which allows the user to control the actual game; the clock, which shows how long the user has spent thinking about each chord; the correct/incorrect screen, which should appear when the user gets it right or presses skip, and should contain visual graphics as well as colours to support my colourblind stakeholder; and finally the current chord, which should show the chord that the user is trying to play as well as if the chord is favourited.

##### Debugging Microphone

Whilst in the creation of the project there are lots of useful graphs and bars that can be plotted to help debug the code and the microphone. The user could use a scaled down version of this to detect whether their microphone is inputting any sound at all.

##### Errors

Another good debugging and helpful feature to the user is a text box to show errors and helpful hints to the user, to make the app easier to use.

#### Design of the UI

The UI will look not dissimilar to this. It should be auto-resizing with the window for ease of use.

A screenshot of a music note

Description automatically generated

##### Help!

The help button’s purpose is if the user does not know the chord, it can display the chord pattern on the screen. This will result in the user getting the question wrong. I put it in the bottom right corner of the screen because it was easy to find but also out of the way and hard to click by accident. I made them blue because it is a contrasting colour that won’t be affected by colour-blindness.

##### Skip

The skip button’s purpose is for the user to skip the chord. I put it in the lower right corner with the help button for similar reasons to the help button. In addition, putting it with the help button is logical because they do similar things. The skip button should move onto the next chord.

##### Exit

The exit button should stop the play function, and in its place, there should be a play button, which should restart the play. This is in the upper right corner because the close button for a window is in the upper right corner, so it is natural for the user.

##### Piano Visual

This should be simplistic and easy to use to find the chord. It should appear and disappear with the help button. I put it at the bottom of the page because it is a long and thin graphic and so is suited to a longer edge like the bottom.

##### Chord

This should be large and contrasting to the background to increase the speed that the user can read it. To do this, I made it white against a grey background, bold, and big.

##### Sheet Music

The sheet music can be automatically generated using a library. I have put it underneath the note because it is the same piece of information displayed in two separate ways.

##### Time

This should be smaller because the user does not need to see it as much as the buttons and chord patterns. It should count from zero with a step of one.

##### Toggle Sheet Music

This button should toggle whether the user sees the sheet music under the note. This is because some stakeholders wanted the sheet music, and some did not. I put it in the lower left hand corner because the user does not need to use it very often.

##### Revision engine

The revision engine will have two main parts to help the user to learn the chords: the file storage, which will ensure that any progress will be kept until the user next opens the app; and the algorithm which works out which chord to send.

This equation will generate a score for each chord:

Where A and B are constants that control how much each variable effects the score. The more time spent on the chord last time or if the chord is favourited means a higher score.

The higher score a chord has, the more chance that it has of being picked. A chord with score 100 will be picked first 100% of the time.

### Usability Features

#### Colour Blind

The app should be just as usable to a colour-blind person as any other person. This affects some aspects of the app, including how a correct/incorrect answer is portrayed, the colours of the foreground and background, and the colours of the writing.

##### App Colours

The foreground and background colours should be contrasting and clear. Because there are types of colour-blindness that affect every colour, the app should use more brighter and darker colours to help people see. For example, because the app could also have a light/dark mode setting, when the app is light, the foreground items should be dark, and when the background is dark, the foreground items should be light.

##### Correct/Incorrect Answer Screen

The answer screen can’t just show a green and red light, because the user might not be able to distinguish between those colours. Therefore, there will need to be symbols like ticks and crosses to denote whether the user has got it right. The original colours can still be used if it has the extra support for colour-blind people.

## Algorithms

### Overall process of the app

A diagram of a process

Description automatically generated

This process involves choosing a chord, which is done by the learning engine, and then displaying that chord. This is followed by checking for a correct input, and moving to the next chord if there is one, and getting the answer wrong if there isn’t one.

I have added a cap for the number of times this loop before the end of the game to make it robust against the user starting and then going away from their keyboard.

## Tests

R = Robustness

F = Functionality

U = Usability

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test Reference | Success Criteria | Description | Test Data | Expected Result | Justification |
| F1.1a | 1.1 | Note is played and a buffer is temporarily written to console | 440Hz note through the microphone | A buffer that isn’t empty (specific buffer values aren’t possible to predict) | This success would mean that the microphone is working, and the program is able to record data |
| R1.1b | 1.1 | The microphone is switched off and the program is run | No noise input | A buffer is not outputted. | There should be no output to the console because the code should not run when there is no input to the buffer |
| F1.2a | 1.2 | Note is played and a graph is plotted of amplitude per frequency. | 440Hz note through the microphone | There is a spike around 440Hz | This would mean that the program can correctly translate from frequency to hertz around the frequencies that are expected. |
| R1.2b | 1.2 | Note is played and a graph is plotted of amplitude per frequency. | 13000Hz note through the microphone | There is a spike around 13000Hz | This would mean that the program can correctly translate from frequency to hertz around even higher frequencies, so it is more robust. |
| F1.3a | 1.3 | Note is played and the note name is displayed | 440Hz note through the microphone | The note “A” is displayed | The program should be able to correctly translate from frequencies to notes |
| R1.3b | 1.3 | Note is played and the note name is displayed | 14080Hz note through the microphone | The note “A” is displayed | The program should be able to correctly translate from frequencies to notes, even at the top of the piano. |
| R1.3c | 1.3 | Note is played and the note name is displayed | 27.5Hz note through the microphone | The note “A” is displayed | The program should be able to correctly translate from frequencies to notes, even at the bottom of the piano. |
| F1.4a | 1.4 | Multiple notes are played separately, and the program can list them | 440Hz Note is played followed by a C note then a Db note | The display shows “A, C” | This would show that the program can handle multiple notes and store them |
| F1.4b | 1.4 | Multiple notes are played together, and the program can list them | 440Hz Note is played followed by a C note then a Db note | The display shows “A, C, Db” | This would show that the program can handle multiple notes at the same time and store them |
| R1.4c | 1.4 | Multiple notes are played together but they are the same | A, A, A is played | The display shows “A” | This would show the program doesn’t count multiple notes twice. |
| U1.5a | 1.5 | Background noise is played in the background | Background noise, e.g. a TV is played at the same time as the notes A and C | The display should show only A and C. | This makes sure that if the user encounters a scenario where they may not have a perfectly quiet environment, then that doesn’t matter. |
| R1.5b | 1.5 | Loud background noise is played in the background | Background noise, e.g. a loud TV is played at the same time as the notes A and C. | The display should show only A and C. | This makes sure that if the user encounters a scenario where someone is screaming at them, they can carry on playing (would not recommend) |
| R1.5c | 1.5 | Notes played that are far out of tune should be counted as background noise | A note of frequency 470Hz is played | This should display nothing | This helps to reduce the interference caused by background noise |
| F2.2a | 2.2 | The user presses start, and a chord appears on the screen. | The user’s previous timings are the test data for this. | The chord displayed should be something that the user is not very good at | This ensures that the user always gets better at a wider range of chords than just the ones they already know |
| F2.2.1a | 2.2.1 | As an extra debugging step, the chords “score” can be displayed. | A time of 10 seconds thinking about an unfavourited chord previously | Should give an output of 0.1A+0.6C where A and C are constants that change how much the time effects the chords. | This test makes sure that the score is working properly so the user is fed different chords |
| F2.2.2a | 2.2.2 | When the user plays another round, then the score should change | After the time of 10seconds, if a time of 7 seconds is produced | The score should output be (0.1A+0.07C)/2 | This test ensures progress is continued throughout the app |
| F2.2.2b | 2.2.2, 2.5 | When the user favourites a chord, the score changes | After the time of 7 seconds, if a time of 4 seconds is produced | The score should output (4A+7C+B(boost))/300 | This test ensures the favourites system works |
| F2.3a | 2.3 | The user plays some rounds and non of the chords repeat | If the first chord is A | The second chord should not be A | Avoiding repeated chord |
| F2.4a | 2.4 | If the user selects “hard” they should see chords that are harder | Hard selected | Harder chord “pack” shown | Advanced users don’t have to start on the easy chords |
| F2.4b | 2.4 | If the user selects “easy” they should see chords that are easier | Easy selected | Easier chord “pack” shown | Advanced users don’t have to start on the easy chords |
| F2.4c | 2.4 | If the user selects “medium” they should see chords that are in the middle | Medium selected | intermediate chord “pack” shown | Advanced users don’t have to start on the easy chords |
| R2.5a | 2.4 | If the user favourites a chord in a pack different to the difficulty they have selected, that should be included | Easy, C#min selected | The user should be fed many chords from easy, plus the C#min chord | This test ensure the user’s favourites are still fed to the user. |
| F3.1a | 3.1 | When the user starts the game, it gives them a chord to play | Start pressed | Chord displayed | So the user can play the game |
| F3.3a | 3.2, 3.3 | the game checks the notes currently being played against the chord notes | On chord “C” being shown  The user inputs C, E, G | The correct screen is shown, the time is reset, and a new chord is shown | This test ensures that the chords can be compared to notes |
| R3.3b | 3.3 | If the notes are incorrect then the chord shouldn’t be played | On chord “C” being shown  The user inputs C, E, F | Nothing happens | This test ensures that the user can’t get the right answer if the press the wrong notes |
| R3.3c | 3.3 | If no notes are played within the time | … | The end of the time should result in an incorrect screen | This test ensures that the game carries on if no notes played |
| F3.5a | 3.5 | If help pressed, answer displayed, user gets it wrong | Help pressed | Answer displayed (chord visual?) | This test ensures that the help button works |
| F3.6a | 3.6 | Whilst in a game the user can favourite a chord | The user favourites a chord mid game | The chord is favourite | This test ensures that the user can favourite a chord in runtime |
| F3.7a | 3.7 | Whilst in a game the user can skip a chord | Skips a chord | Time of 60s added, next chord displayed | This test ensures that the skip button works |
| F3.8a | 3.8 | At the end of the time limit, the answer is “incorrect” | User doesn’t input | The answer is incorrect at end of time | Moves the game on |
| F4.1a | 4.1 | Scalability of the UI | Small UI | All controls fit and the UI is still legible | Evaluating the resizing of the UI |
| F4.1b | 4.1 | Scalability of the UI | Medium UI | All controls fit and the UI is still legible | Evaluating the resizing of the UI |
| F4.1c | 4.1 | Scalability of the UI | Large UI | All controls fit and the UI is still legible, and they fill the space as best as possible. | Evaluating the resizing of the UI |
| F4.2a | 4.2 | When a user gets more than three right in a row, a streak count should be present. | User’s input is three right in a row | “3” | Testing that the streaks works |
| F4.4a | 4.4 | The user should be able to see whether the chord is favourite or not whilst playing the game | User favourites “C” then plays the game | An indicator is displayed next to “C” | This test ensures the favourited system works |
| F4.5a | 4.5 | Background changes to green when correct | User gets an answer correct | The background changes to green with ticks | This ensures that the user can check their answer |
| F4.5b | 4.5 | Red when incorrect | User presses help | Red screen shown | When wrong, red screen |
| F4.5c | 4.5 | When timer runs out, red screen | No input after game start | Red screen shown | When the user runs out of time the get it wrong |
| U4.6a | 4.6 | There are ticks and crosses on the screen | For any of 4.5 | Crosses shown/ticks shown as well as colour | Makes sure that colourblind settings work |
| F4.7a | 4.7, 4.8 | When the help button is pressed the incorrect sequence should play and a keyboard graphic should appear on screen with the answer | “Help” button pressed | Incorrect screen shown and graphic with the chord name shows | This makes sure that the help button is working |
| F4.8a | 4.8 | When the button for sheet music is toggled, the sheet music for the chord is displayed | Game started; sheet music toggled on | Shown sheet music for the chord shown | This makes sure the sheet music is working |
| R5.1a | 5.1 | No microphone selected | When the user starts a game without the microphone | A pop-up should appear “microphone strongly recommended” | This helps the app to cope with no microphone |
| R5.1b | 5.1 | No sound from selected microphone | When game starts, if there is no sound | A pop-up should appear “no noise detected from selected microphone” | This helps the app to cope with no microphone |
| R5.2a | 5.2 | Badly Tuned piano – does its best but doesn’t respond to notes that are too far away from the accurate frequency | If the frequencies played aren’t notes | Ignores the frequencies | This is so that there is less chance of an error |
| R5.4a | 5.4 | Notes that are too loud or quiet | The note is really loud | Counted as a normal note. Suggestion that the note is too loud | There is less chance of note error |
| R5.4b | 5.4 | Notes that are too quiet | The note is really quiet | Suggestion that the note is too quiet | So there is less chance of error |

# Development and testing

## UI

The first step is to create a UI that is scalable and user friendly.

To do this, I am going to use windows forms in visual studio’s drag-and-drop app.

### Relevant code/design:

A screenshot of a computer

Description automatically generated

### Test F4.1a:

A screenshot of a computer

Description automatically generated

This passes the test because in this small window, all the controls are legible and clickable.

### Test F4.1b:

A screenshot of a computer

Description automatically generated

This passes the test because in this normal sized window, all the controls are legible and clickable.

### Test F4.1c:

A screenshot of a computer

Description automatically generated

This passes the test because in this large window, all the controls are legible and clickable.

## ChordPack Structure

I then created the classs “Chord”, and “ChordPack”.

A screenshot of a computer

Description automatically generated “Chord’s” attributes were: a list of notes, implemented as letters; a score for competance; and the chord name. The constructor is structured so that you can add 0-5 notes upon construction, and there is a separate method for this. This method, “addNote”, contains some validation to check that its input isn’t null, because I was running into an error where it was trying to add nothing to the list if the chord contained less than 5 notes.

ChordPack’s attributes are a list of chords and the name of the pack. And you specify these on construction. It has methods: AddChord, to add a chord to the pack; GetChords, to get the list of Chords in the pack without being able to edit it; and an override for ToString, which just returns some details about the pack.

A screenshot of a computer program

Description automatically generated

## Chord Loading

Next, I implemented loading chord packs from txt files. This involved the class “Revision Engine” having a function “loadChords” which created a “ChordPack” object from a txt file by using Regex to match each “Chord”.

A screen shot of a computer program

Description automatically generated

This code does not run without crashing:

A screenshot of a computer program

Description automatically generated

This error is caused by the List of chords in a pack being null. To fix this, I can add an overload for the chord pack constructor that just creates a blank list if there are no chords to specify yet. A screen shot of a computer program

Description automatically generated

Then this code runs without crashing, and if I write the chord packs to console, the information is correct to the txt file.

A black background with white letters

Description automatically generated



## Displaying the chord

To assess this, I have set up a random chord chooser (an algorithm to set up confidence and score will be implemented later):

A computer screen shot of text

Description automatically generated

### Test F2.2a

**A screenshot of a computer

Description automatically generated**

Test pass – chord appeared on start pressed.

### F2.2.2a

A screenshot of a computer

Description automatically generated

Test pass - chord appeared when re-clicked start.

### F2.3a

Test fail – the chords repeated after clicking start twice. To fix, I need to implement a memory.

A computer screen shot of a program code

Description automatically generated

This code had a runtime error:

A screen shot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

I fixed it by changing the random number values to dynamically change to the size of the chord.

The code then passed the test as all the Chords displayed were different.

## Listener

A computer screen shot of a program code

Description automatically generatedA computer screen shot of text

Description automatically generatedA screen shot of a computer program

Description automatically generatedA computer screen shot of a program code

Description automatically generatedA computer screen shot of a program code

Description automatically generatedA computer screen shot of code

Description automatically generated

I wrote the code for the listener methods that I had mostly planned out, with some exceptions, highlighted below.

### F1.1a

A screen shot of a computer

Description automatically generated

This test is passed because the buffer is writing to console, and the buffer is not null. This is the code at this stage that makes this test happen: A screen shot of a computer program

Description automatically generated

### R1.1b

This is easily implemented into the last test:

A black background with colorful text

Description automatically generated

When I unplug my microphone, and run it, there is no buffer written and null is returned. This means that the test has passed.

### F1.2a

A graph of a microphone

Description automatically generated

Test passed because the spike is around the right place, and there is a spike.

### R1.2b

A graph of a microphone

Description automatically generated

Test passed because there is a spike in the correct place on the graph.

### F1.3a



Test passed.

### R1.3b



Test passed.

### R1.3c



Test passed.

### F1.4a



Test passed.

### F1.4b



Test passed.

### R1.4c

(A4 and A5 played)

Output:



### U1.5a, b



Test passed.

Evaluation

1. https://visualstudio.microsoft.com/ [↑](#footnote-ref-1)
2. Adapted from information at https://en.wikipedia.org/wiki/Pulse-code\_modulation. [↑](#footnote-ref-2)
3. Adapted from information at https://en.wikipedia.org/wiki/Fast\_Fourier\_transform. [↑](#footnote-ref-3)