€

Candidate Number: 7629

Jonathan Broster

A windows app that gamifies the process of learning piano chords

H446

A-Level Computer Science

A-Level Computer Science

Practical Programming Project

Contents

[1 Analysis 3](#_Toc159526949)

[1.1 Summary 3](#_Toc159526950)

[1.2 Further Problem Identification 3](#_Toc159526951)

[1.3 Computation 3](#_Toc159526952)

[1.4 Stakeholders 4](#_Toc159526953)

[1.5 Interviews 4](#_Toc159526954)

[1.6 Research 7](#_Toc159526955)

[1.7 Essential features 9](#_Toc159526956)

[1.8 Limitations 10](#_Toc159526957)

[1.9 Optional Features 10](#_Toc159526958)

[1.10 Requirements 11](#_Toc159526959)

[1.11 Feature List 11](#_Toc159526960)

[2 Design 17](#_Toc159526961)

[2.1 Decomposition 17](#_Toc159526962)

[2.2 Structure of the Solution 18](#_Toc159526963)

[2.3 Algorithms 27](#_Toc159526964)

[2.4 Tests 27](#_Toc159526965)

[3 Development and testing 36](#_Toc159526966)

[3.1 Process 36](#_Toc159526967)

[3.2 Prototype 1 36](#_Toc159526968)

[3.3 Prototype 2 69](#_Toc159526969)

[Evaluation 90](#_Toc159526970)

[3.4 Post-development testing 91](#_Toc159526971)

[3.5 A close-up of black text

Description automatically generated 91](#_Toc159526972)

[3.6 Testing Evaluation 91](#_Toc159526973)

[3.7  91](#_Toc159526974)

[3.8 Unmet Success Criteria 91](#_Toc159526975)

[3.9 A close up of black text

Description automatically generated 91](#_Toc159526976)

[3.10 Usability Evaluation 91](#_Toc159526977)

[3.11 Future Development of Unmet Success Criteria 91](#_Toc159526978)

[3.12 Maintenance 91](#_Toc159526979)

[3.13 Future Improvements 91](#_Toc159526980)

# 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 use 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

These are features that need to be included in the app for it to function as intended.

### 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

These are features that would be nice to have in the app, and won’t necessarily be a priority.

### 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)

## Requirement Specification / Feature List

|  |  |  |
| --- | --- | --- |
| Requirement Number | Description | Justification |
| 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. |
| 6 | Saving user data |  |
| 6.1 | The data should be automatically saved after each round | This is because the user should be able to carry on from where they left off. |
| 6.2 | The data should be loaded from a local disk when the game starts. | This is because the user should be able to carry on from where they left off. |

# 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.

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 (Feature 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() |

#### Constructor

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 two 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. The following sections describe each part 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 | Feature | Description | Test Data/Process | 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 with a C note then a Db note | The display shows A C E | 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 | The user selects chord packs from the list and they are used in the game | Chord pack 1 & 2 selected | These chord packs are used in the game | So the user can select chord packs to learn. |
| F2.4b | 2.4 | The user can make new chord packs, and as long as they are in the correct format, they should work with the game. (No rigorous validation required as it is up to the user to provide the ChordPacks in the correct format) | A chord pack is added Containing different chords | These chord packs are displayed in the list | So the user can select chord packs to learn. |
| R2.4c | 2.4 | A small amount of validation is required to stop the game from crashing if the user doesn’t format a pack correctly, but not to check they got the chords in the correct order. | A chord pack containing some invalid data is placed into the chord pack folder. | This pack is ignored. | So the user can’t crash the program or input code. |
| R2.4d | 2.4 | Check that the chords are in a correct format. | A chord pack containing data in the wrong format is placed into the folder. | A notification appears saying that the data isn’t formatted correctly | This test ensures that the user can’t crash the program or code. |
| R2.5a | 2.5 | 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. |
| F2.6a | 2.6 | The user should be able to close and reopen the app and all their progress (chord scores) should be saved | User plays the game, then closes the app | User plays the game again with the same chord packs and the scores are the same as what they were before the app closed | This test makes the data saveable |
| 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 |
| F3.6a | 3.6 | Whilst in a game the user can favourite a chord and it stays favourited until it is unfavourited. | The user presses favourite on AbMaj chord. | The chord is favourite, and when the chord is seen again it is still favourite. | This test ensures that favouriting a chord actually stays favourited and the UI keeps up with that |
| 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 | There should be a selection for the packs that are available | User presses start | Packs shown and selectable | Makes it easier for the user to select the packs |
| R4.7b | 4.7 | No chord packs selected | The user doesn’t select any chord packs | It should prompt a chord pack choice | This is a validation step to ensure the program doesn’t crash |
| F4.7c | 4.7 | The user selects a chord pack it should be playable | The user selects “Major – beginner” | When start is pressed, should only see chords from the major chord pack. | This is so that if the user selects a chord pack it actually shows chords to them. |
| F4.7d | 4.7 | If a chord pack is unchecked after it is checked it should be removed from the list | The user selects a pack, then plays, then deselects a pack, then plays | It should update the packs in the game accordingly | This is to ensure that the user can change the packs chosen in between play. |
| F4.7e | 4.7 | Any chord packs that are listed in the folder should appear in the table | The user adds a chord pack to the folder called “Minor - beginner” | It appears in the table and is selectable | This is so that the user can download new chord packs, making their own - unrecommended |
| F4.8a | 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 |
| F4.9a | 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 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 quiet | Suggestion that the note is too quiet | So there is less chance of error |

# Development and testing

Please note that code in development may not be commented. See the end of the section for completed and commented code.  
Please also note that on each iteration, every test code was re-tested, but only the relevant tests to the changes made in the code are in the report.

The development of this project will have an agile methodology and I will aim to meet certain milestones and prototypes in the project. Here is a Gantt Chart outlining the process of development and timings, as well as the milestones and iterations that I will do. It is broken down into the modules of the topic, and each iteration/version of a module’s development shall be the prototypes.



Each yellow rhombus represents a milestone in the development, and at each milestone, there will be testing and client feedback.

I am using this process because it allows me to keep checking in with the client so that I know that the project will always be exactly what they want, and any issues can be fixed in the development. Another benefit of using this kind of methodology is that I can see very clearly how much time I should spend on each of the sections of the project, and I can keep track of where I am in the development very easily.

## Prototype 1

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.

### UI - First Iteration

A screenshot of a computer

Description automatically generated

This is a picture of the first iteration of the design of the UI. I have implemented a table layout so that the buttons are all the same size, and they can be scaled with the window. At this stage, none of the buttons do anything and the text doesn’t change, but I have the fundamentals of the UI so I can implement the back end later. The title and text font I chose is Roboto because of its tidy and smooth design, and its legibility. In addition, I chose the font sizes so that none of the elements particularly overpower the app.

To further address the layout and the size of the elements, I implemented a visual hierarchy. The elements should be seen in this order:

|  |  |  |
| --- | --- | --- |
| Number | Element | Justification |
| 1 | Chord – Question | Because the user will be looking quickly up and down at the piano it is important that they can easily find the correct part of the screen to look at. |
| 2 | Title | This is an important visual for the user because it can be used to identify the app. |
| 3 | Start button | This is important because it is needed to play the app, and it’s one of the main buttons. |
| 4 | Everything else | For the rest of the elements, it doesn’t really matter what order they are in, because they do not need to stand out. They are elements that the user will seek out rather than have to be prompted to press. |

There are multiple ways of implementing a visual hierarchy ([[4]](#footnote-4)):

* Size – Users notice larger elements more easily.
  + I need to make the Chord and title larger than the rest of the elements.
* Colour – Bright colours typically attract more attention than muted ones.
  + I need to make the chord brighter than the surrounding buttons.
* Contrast – Dramatically contrasted colours are more eye-catching.
  + Need to make the contrast of the chord more than the rest of them.
* Alignment – Out-of-alignment elements stand out over aligned ones.
  + Because the chord is centred and the rest of the elements are in alignment around the edge, this helps here.
* Repetition – Repeating styles can suggest content is related.
  + I can’t really implement this here.
* Proximity – Closely placed elements seem related.
  + The help and skip buttons are close together.
* Whitespace – More space around elements draws the eye towards them.
  + There is lots of space around the chord and title.
* Texture and Style – Richer textures stand out over flat ones.
  + Everything has similar textures, except the chord and title which are bolder.

In the first iteration, I am trying to figure out where to position the elements to account of this visual hierarchy, as well as the scalable UI.

### UI - First Iteration Tests

|  |  |  |  |
| --- | --- | --- | --- |
| Test Number | Description | Code | Evaluation |
| F4.1a | Scalability of the UI | A screenshot of a computer  Description automatically generated | In this small window, all of the buttons fill the space and all of the writing is clear and legible. However, there is a problem with the icon in the bottom left being squashed due to the form factor.  FAIL |
| F4.1b | Scalability of the UI | A screenshot of a computer  Description automatically generated | In this medium window, all of the buttons fit into the space and the writing is legible.  PASS |
| F4.1c | Scalability of the UI | A screenshot of a computer  Description automatically generated | In this large UI, the text is quite small and the buttons don’t take up enough space. The icon in the bottom left is stretched.  FAIL |
| F4.2a | When a user gets more than three right in a row, a streak count should be present. | A screenshot of a computer  Description automatically generated | FAIL – no streak indicator present |
| F4.4a | The user should be able to see whether the chord is favourite or not whilst playing the game | A screenshot of a computer  Description automatically generated | FAIL – no favourite indicator present |

These tests mostly failed which shows that the UI at the end of this section needs another iteration. The main focus points for the next iteration will be: Scalability improvements, and ensuring everything fits in the screen no matter what size the screen is; A streak indicator, which should give a more game-like feel; a multitude of other features which were overlooked in this iteration

### UI – Second Iteration

I decided to do UI 2 out of order because a lot of features depend on the UI being suitable and because I need an output to the user to test and develop the features of the other modules.

A screenshot of a computer

Description automatically generated

This is a picture of the relevant code and design for this iteration. In this iteration I added a favourite indicator and button (heart bottom left), streak indicator (fire, right), packs selector, (left). I also fixed bugs with stretching images and added a correct/incorrect screen.

It is important to add these images to add some colour and make the app more visually appealing, to increase retention time.

I also added some basic functionality to these buttons and labels, such as for the like button, on click the heart change from full to empty and vice versa, or the streak update function to run on correct answer.

### UI - Second Iteration Tests

|  |  |  |  |
| --- | --- | --- | --- |
| Test Number | Description | Code | Evaluation |
| F4.1a | Scalability of the UI | A screenshot of a computer  Description automatically generated | In this small window, all of the buttons fill the space and all of the writing is clear and legible.  PASS |
| F4.1b | Scalability of the UI | A screenshot of a computer  Description automatically generated | In this medium window, all of the buttons fit into the space and the writing is legible.  PASS |
| F4.1c | Scalability of the UI | A white background with black letters  Description automatically generated | In this large UI, The text is a legible size and the buttons feel less spread out.  PASS |
| F4.2a | When a user gets more than three right in a row, a streak count should be present. | A screenshot of a computer  Description automatically generated    This code runs on correct answer    This code runs on incorrect answer | PASS – streak indicator present and functional |
| F4.4a | The user should be able to see whether the chord is favourite or not whilst playing the game | A screenshot of a computer  Description automatically generated  A screen shot of a computer code  Description automatically generated  This code runs on heart click and toggles the fill of the heart as an indicator. | PASS - favourite indicator present |
| F4.5a, b | Background changes to green when correct | A screen shot of a computer program  Description automatically generated  This piece of code makes the screen red and green when the questions are wrong or right. The screens look like this:  A screenshot of a phone  Description automatically generated  A screenshot of a phone  Description automatically generated | PASS - The background changes to green with ticks.  These screens have green and red to be visually striking, but also tick/cross so that colour blind users aren’t disadvantaged. |
| F4.5c | When timer runs out, red screen | A red x in a square with black text  Description automatically generated | PASS – Red screen shown when no answer present |
| U4.6a | There are ticks and crosses on the screen | A red x in a square with black text  Description automatically generatedA screenshot of a phone  Description automatically generated | PASS – there a re ticks and crosses so that it is accessible to colour-blind people. |
| F4.7a | There should be a selection of available packs | A screenshot of a computer  Description automatically generated  A screen shot of a computer code  Description automatically generated  This bit of code makes a list of the indexes of packs that are checked.  A black screen with white text  Description automatically generated  This bit of code passes the list of pack indexes to the revision engine so that it knows what to base the next chords from. | PASS – the user can select a chord pack and it adds it to the possible chords. |
| R4.7b | Check if the user hasn’t selected any chord packs | A screenshot of a computer  Description automatically generated  A screenshot of a computer program  Description automatically generated | PASS – the program gives a warning dialogue box if there are no chord packs selected. |
| F4.7c | The user selects a chord pack it should be playable | The user selects “Major – beginner”.  A blue and white sign with black text  Description automatically generated  A black and white image of a letter  Description automatically generated  A black letter with a white background  Description automatically generated A black letter with a white background  Description automatically generated A black letter on a white background  Description automatically generated  These are the expected chords, as these are the chods contained in this chord pack.  A screenshot of a computer screen  Description automatically generated | PASS – only the chords from the major pack are showed. |
| F4.7d | If a chord pack is unchecked after it is checked it should be removed from the list | A screenshot of a computer  Description automatically generated  A screen shot of a computer code  Description automatically generated  I fixed this issue so that on start it resets the list of packs instead of just adding to it. | PASS – when a chord pack is unchecked it is not in the list, even if it previously was. |
| F4.7e | Any chord packs that are listed in the folder should appear in the table | A screenshot of a computer  Description automatically generated | PASS – all the chord packs listed are available for selection |
| F4.8a | When the help button is pressed the incorrect sequence should play and a keyboard graphic should appear on screen with the answer |  | FAIL – kind of – there is a button but it does nothing yet as it has not been implemented. |
| F4.9a | Toggle sheet music button – onclick, it should display music |  | FAIL – there is a button but it does nothing yet as it has not been implemented |

UI iteration 2 conclusion – Failed tests 4.8 and 4.9 are yet to be implemented.

### Revision Engine First Iteration

The first iteration will focus on the development of chords and loading chordpacks.

#### ChordPack Structure

I then created the class “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 creates a “ChordPack” object from a txt file by using Regex to match each “Chord”.

A screen shot of a computer program

Description automatically generated

A screen shot of a computer program

Description automatically generated

I can add an overload for the chord pack constructor that just creates a blank list if there are no chords to specify yet.

Then 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



### Revision Engine First Iteration Tests

|  |  |  |  |
| --- | --- | --- | --- |
| F2.2a | The user presses start, and a chord appears on the screen. | This code runs on the start button press, and first makes a list of the chord packs that are selected and then calls NewQuestion,    Which first reset all the timers and screens and then sets the chord control test to the output of the method NextChord,    Which makes a list of the possible chords out of the list of possible chord packs, and then removes the previous chord from the list as to avoid a situation where the same chord is played twice. Then, it chooses a chord using a random function (temporary) and returns that chord. | PASS – this successfully displays a new chord when the user presses start. |
| F2.2.1a | As an extra debugging step, the chords “score” can be displayed. |  | FAIL – the score is not displayed. (not yet implemented) |
| F2.2.2a | When the user plays another round, then the score should change |  | FAIL – the score is not displayed. (not yet implemented) |
| F2.2.2b | When the user favourites a chord, the score changes |  | FAIL – the score is not displayed. (not yet implemented) |
| F2.3a | The user plays some rounds and none of the chords repeat | This code makes sure the previous chord played is not part of the possible chords for the next question. | PASS - I skipped through a bunch of chords and found that there are never any repeats. |
| F2.4a | The user selects chord packs from the list, and they are used in the game | This code is run on start pressed, and compiles a list of chords selected which it gives to the NextChord func.    This code adds the chosen packs to the possible chords. | PASS – The code successfully gives the user any packs from the chords that they have suggested. |
| F2.4b | The user can make new chord packs, and if they are in the correct format, they should work with the game. (No rigorous validation required as it is up to the user to provide the Chord Packs in the correct format) | The Pack I made:      It appears in the list:    And it is playable: | PASS – the user can create their own packs in the correct format and they are playable. |
| R2.4c | A chord pack containing some invalid data is placed into the chord pack folder. And a suitable error message is displayed. | I added this blank chord pack into the folder.  Relevant code:    (loadchords returns null if there isn’t a match found by the regex) | PASS – Displays an error message if the chord pack is empty |
| R2.4d | A chord pack containing data in the wrong format is placed into the folder. A suitable notification appears | This is the chord pack that put in with invalid data: |  |
| R2.5a | If the user favourites a chord in a pack different to the difficulty they have selected, that should be included |  | PASS – the correct chord pack is displayed |

In this iteration, everything passed except for F2.2 – the score to sort the chords, because of not yet implemented.

### Listener First Iteration

I used <https://www.youtube.com/watch?v=qUlCImYOC8c> (Scott Harden (ScottPlot) on real-time Fourier transforms) to help understand some of the complex maths behind the calculations.

For more information on this class, see the design section. A computer screen shot of a program code

Description automatically generatedA computer screen shot of text

Description automatically generated

The following code was somewhat adapted from Scott Harden’s video:A screen shot of a computer program

Description automatically generatedA computer screen shot of a program code

Description automatically generated

This FFT method ends the adapted code.A computer screen shot of a program code

Description automatically generatedA computer screen shot of code

Description automatically generated

For the PullNotes method, I changed it slightly from what I suggested in the design. These changes made the notes more accurate. For example, discarding any notes that are too far away from an actual note in the WhatNoteAmI method, and instead returning null, meant that I had to add a statement to check if the note returned was null.

I tested this listener class on my laptop and found that some of the notes were not being register. I realised that the microphone’s sensitivity was different, and that it would be different on every device. I was also then met with a problem where the listener would detect not just the base note but also all of the harmonics underneath it, as for this test I was using a real piano, not just frequency generators.

To fix this, I added a dynamic threshold to the PullNotes method:



The method Calibrate:

A computer screen with white text

Description automatically generated

This method takes the highest amplitude frequency in the array and returns that (minus an arbitrary number) as the threshold. To work out this arbitrary number, I used a FFT graph of a known chord and visually identified the peaks and troughs (and therefore the base note) and looked at the difference between the base note’s amplitude and the harmonics that are found.

### Listener First Iteration Tests

To test these I will use <https://muted.io/piano-chords/> which allow me to play any chords through my microphone by putting the microphone next to a speaker. I will also use <https://onlinetonegenerator.com/> to generate specific notes

|  |  |  |  |
| --- | --- | --- | --- |
| Test Reference | Description | Evidence/Relevant code | Justification |
| F1.1a | Note is played and a buffer is temporarily written to console | A screen shot of a computer  Description automatically generated A screen shot of a computer program  Description automatically generated | PASS - 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: |
| R1.1b | The microphone is switched off and the program is run | A black background with colorful text  Description automatically generated | PASS - 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 | Note is played and a graph is plotted of amplitude per frequency. | A graph of a microphone  Description automatically generated | PASS - Test passed because the spike is around the right place, and there is a spike. |
| R1.2b | Note is played and a graph is plotted of amplitude per frequency. | A graph of a microphone  Description automatically generated | PASS - Test passed because there is a spike in the correct place on the graph. |
| F1.3a | Note is played and the note name is displayed | when  was played | PASS – the test is passed because an “A” note is dispayed |
| R1.3b | Note is played and the note name is displayed | when  was played |  |
| F1.4a | Multiple notes are played separately, and the program can list them | when  Pressed “A” the “C” |  |
| F1.4b | Multiple notes are played together, and the program can list them | Was outputted when A C and E were pressed at the same time | PASS 50% - Although it did display the correct notes, because the notes are made up of lots of smaller notes, there were some extra notes played |
| R1.4c | Multiple notes are played together but they are the same | Was outputted when A was pressed 3 times | PASS 50% - It correctly filtered out the repeated note but also thought the note A was made up of 3 different notes. |
| U1.5a | Background noise is played in the background | Was outputted when a and C where played at the same time as background noise | PASS – This test passes because no extra notes were outputted |
| R1.5b | Loud background noise is played in the background | Was outputted When drilling noise was played at same time as ntoes | FAIL – because the notes were not the ones expected and there were additional notes, it fails this test. |
| R1.5c | Notes played that are far out of tune should be counted as background noise | Was played and  was outputted | FAIL – notes were outputted not ignored |

In conclusion, the listener functions correctly, but the accuracy could be improved. For example, some notes were outputted even if they were not played, and some notes weren’t outputted when they were played.

### Game First Iteration

Next, I began to code the game. I made a few basic procedures that could help with this:



This procedure resets the game to the start point.



This procedure provides a new question for the user. It first resets the timer etc., then displays a chord.

A computer screen shot of a program

Description automatically generated

These two procedures are called when the user gets it wrong or right. They display a tick or cross then change the streak.

A computer screen with white text

Description automatically generated

This procedure removes the tick or cross placed by the Correct/Incorrect Screen procedures.

With these procedures in mind, I added the \_Click ability to the buttons:

When skip button pressed:

A screen shot of a computer code

Description automatically generated

On start button clicked, a few things need to happen.

A screen shot of a computer program

Description automatically generated

If it’s a stop button that’s pressed, disable times, and stop the mic listening, and reset. (change back to start too)

Start the mic, put a new question out and change the button text to stop.

If the user has not selected a pack, display a message reminding them to do so.

Iterate through the list of checked packs and make a list of the indexes checked to pass into the next chord method.

I could then implement the two timers that I need: Countdown timer, for the clock; and listenTick timer, so the listener knows when to listen.



A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated



This countdown timer should decrease the clock on tick and if the clock is less than 0, it should trigger an incorrect question.

A screen shot of a computer program

Description automatically generated

A lot needs to happen on this regular 20ms tick:

* Disable the timer whilst any calculations happen
* Create a list of notes played from the microphone
* Check that there are notes
* Check if the notes are correct for the current chord
* Trigger either correct or incorrect
* Re-enable the timer

### Game First Iteration Tests

|  |  |  |  |
| --- | --- | --- | --- |
| F3.1a | When the user starts the game, it gives them a chord to play |  | PASS – The app displayed a chord in the centre of the screen when I pressed “start” |
| F3.3a | the game checks the notes currently being played against the chord notes |  | PASS – The game displays all then notes it hears and checks against the chord. |
| R3.3b | If the notes are incorrect then the chord shouldn’t change |  | PASS – the app doesn’t change the chord when notes are played which are not in the chord |
| R3.3c | If no notes are played within the time the answer is classed as wrong |  | PASS – The time ran out and this was displayed, and it moved onto the next chord. |
| F3.5a | If skip pressed, answer displayed, user gets it wrong |  | PASS – 50% skip button pressed gives incorrect answer and moves onto next chord.  Did not display the correct answer |
| F3.6a | Whilst in a game the user can favourite a chord |  | PASS – heart button pressed and chord got favourite status |
| F3.6a | Whilst in a game the user can favourite a chord and it stays favourited until it is unfavourited. | I favourited A MAJ:    Amaj was favourited the next time it came to it | PASS – heart button pressed and chord got favourite status, and retains it until the next round |
| F3.7a | Whilst in a game the user can skip a chord |  | PASS – I pressed skip, and it gave the incorrect markings and moved onto the next chord |
| F3.8a | At the end of the time limit, the answer is “incorrect” |  | PASS – The time ran out and this was displayed, and it moved onto the next chord. |

## Prototype 1 Development Summary & Evaluation

At the end of this iteration, I tagged the commit “V1” on GitHub and this was the commit that my user’s reviewed.

A screenshot of a computer program

Description automatically generated

### Client Feedback

I gave my app to a user and recorded their experience of it. I made sure to note what order they did things, if they struggled with anything, etc. After talking with the user, I identified contiguous points of feedback, listed in the table below.

|  |  |  |  |
| --- | --- | --- | --- |
| Code | Feedback | Module | Analysis |
| 1 | The user was confused when first faced with the app. They struggled to identify how to start the game and how to play. | UI | This feedback shows that the UI needs to assist the user more in learning how to play the game. The user’s eye’s need to be drawn to the way forward. This is key to the functionality of the app and this feedback means that the app is unusable to a stranger at this version. |
| 2 | Overall, the writing was too small for the user to read easily without squinting. | UI | This feedback shows that the UI needs to be changed so that the text is larger and clearer. It is not critical to the functionality of the app, but it is a key usability feature. |
| 3 | In the list box for selecting the chords, the user had trouble determining that it was a double click for the selection, and advised that they would prefer if it wasn’t | UI | This feedback shows that the list box should be updated to allow for check on click. It is not critical to the functionality of the app, but it is a key usability feature. |
| 4 | The user was confused by the notation of some of the chords, notably, the major chords, because they are used to seeing the shortened “A” on sheet music to mean A major, rather than Amaj | Chord Packs | This feedback shows that the naming of the chords should be investigated, and a decision should be made regarding the convention. It is not critical to the functionality of the app, but it is a key usability feature. |
| 5 | The user experienced roughly 7/30 times that a chord would give a correct tick and move on before the user had played all the notes in the chord. | Listener | This feedback shows that the listener needs to improve at identifying notes, as sometimes there was only one note being played and it thought there was two or more. |
| 6 | When the user would get to a Major 7th chord e.g., “C7” they would play the correct chord and it would not be correct. | Chord Packs | This feedback indicates a bug in the musical notation or interpretation of the major chords pack. It is a critical function of the app, but it is a key usability feature. |
| 7 | The user commented that the game didn’t feel very “gamey” and requested more user interactions, for example messages of encouragement, more emphasis on the streak, more transparent analysis of their performance, etc. | Game | This feedback shows that the user expected more of a game than what they received, which should be a particularly crucial factor as it was set in the success criteria. This is critical to the user experience of the app. |
| 8 | The user requested that there be more emphasis on the time, to generate more stress and more explanation of what will happen if the time runs out | UI | This feedback also shows that the user expected more of a game than what they received, which should be a particularly crucial factor as it was set in the success criteria. |
| 9 | The user also stated the desire for a timeout for each note they played, so after a certain amount of time the computer forgets the note and the have to play it again. | Listener | This feedback shows that the notes should not last forever and the user wants to have to re-play them to get the correct mark. |
| 10 | The user wanted to close the app and pick up where they left off, but their progress was reset | Revision Engine | This means I need to implement the progress saving |

### Testing Evaluation

The first set of tests run was tests 4-, which links to the UI. After UI 1, the tests mostly failed which shows that the UI at the end of the section needed another iteration. The focus points for the next iteration were scalability improvements, and ensuring everything fits in the screen no matter what size the screen is; a streak indicator, which should give a more game-like feel; a multitude of other features which were overlooked in this iteration. In UI 2, the scalability was fixed, a streak was implemented, and other features note included in UI 1 were introduced such as: favourite chord indicator, background showing ticks/crosses at the appropriate places, pack selector, help and skip buttons. The only test that didn’t pass was the implementation of the help button (F4.8a). This should be implemented in the next iteration.

In the revision engine module, I implemented the selection of chords, and added logic to ensure they don’t repeat, and that it is only selected from chord packs that are selected by the user. Also, I implemented the loading of custom chord packs from a file, and validation to check if the chord pack is in the correct format. Overall, these tests all passed, except the tests that referenced a score, which has not yet been implemented. This means that in the next iteration, I will need to implement the score.

The next stage of development was the listener module, which some tests ran well, and some tests ran inconsistently. For example, in F1.4b&c, although it did display the correct notes, because the notes are made up of lots of smaller notes (harmonies), there were some extra notes played. Also, in R1.5b&c it failed to get rid of any load background noise to distinguish the notes and failed to ignore notes far out of the range. This means I need to refine the methods of selecting which frequencies are notes in the next iteration.

### Evaluation

Overall, there are some problems with most modules that were highlighted in both the tests and the user feedback. Some of these problems are smaller and don’t impact the success criteria much, whilst others are larger and impede the use of the app.

In the UI module, most of the tests passed, but it was highlighted in the client feedback that there are a number of issues regarding the size of text, and explanations about how to use the app. These are especially important to have and should be implemented in the next iteration, along with the client’s request of making it more like a game, and therefore making it more stressful but fun.

In the Chord Packs/Revision Engine module, the notations of some of the chords need to be looked at and changed for example, the major chords can be just represented by a letter, and the major seventh chords didn’t say major seventh in the names. This is a problem with the chord packs that are provided with the app, and therefore needs to be investigated.

In the listener module, both the user and tests indicated that there is a bug where more than one note will be detected when only one note is played. The user reported this happening about 20% of the time, and the tests reflected this. In the next prototype, there needs to be an attempt to rectify this.

In the game module, whilst the tests ran fine, the user requested a more “gamey” experience, and more transparent scores and streaks, etc.

However, there were also parts that user and the tests agreed were good, for example, the selection of chords and the ability to add your own chords.

## Prototype 2

In this prototype I aim to fix bugs found in both the user error and testing tables in the last prototype, whilst not failing any tests that were previously passes.

### Revision Engine Second Iteration

In this iteration, I will focus on fixing the tests that didn’t work. This will involve implementing a score:

First, I need to add an attribute “score” to the chord class:



Then, I can make a method in Revision Engine to calculate the chord score and save it to the chord.

A computer screen shot of a black screen

Description automatically generated

This can then be called when the answer is correct or incorrect.

A screenshot of a computer

Description automatically generated

A screen shot of a computer

Description automatically generated

I can then add some code to choose the chord based on the score it has:

A black background with white text

Description automatically generated

The user feedback also suggests that I should implement a save file:

I added a method in Revision Engine that saves the user data:



This method requires the use of the NuGet Package “CSVHelper”. The method opens a csv file and using a mapping of the class of the chord, writes the list of chords to a file, using the chord’s attributes as headers.

A screenshot of a computer

Description automatically generated

This was outputted when the function was ran:

A screenshot of a computer

Description automatically generated

Which shows that the function works.

I now need to implement a loadUserData function:

A computer screen shot of a program

Description automatically generated

This function opens the csv, loads the chords stored in it, then puts the values in the loaded chords into the values in the in-game chords. This function uses a helper function “FindChords”, displayed below:

A computer screen shot of a program

Description automatically generated

This function returns all the chords in the “allChords” list with a specific name.

I also need to edit the default chord packs as the user feedback stated.

A screenshot of a computer screen

Description automatically generated

Turned the Xmaj -> X

A black background with white text

Description automatically generated

Turned X7 -> Xmaj7

### Revision Engine Second Iteration Tests

|  |  |  |  |
| --- | --- | --- | --- |
| F2.2.1a | As an extra debugging step, the chords “score” can be displayed. | A screen shot of a computer  Description automatically generated  This function calculates the score for each chord using the function in the Design section.  A black background with white text  Description automatically generated | PASS – the score is displayed. |
| F2.2.2a | When the user plays another round, then the score should change | A computer screen shot of a program  Description automatically generated  This code reorders the list of chords so that the worst chords for the user are displayed. | PASS – the score is different to last time |
| F2.2.2b | When the user favourites a chord, the score changes | A screen shot of a computer program  Description automatically generated  This code favourites a chord which is included in the calculation. | PASS – The score changes when favourited) |
| F2.6a | The user should be able to close and reopen the app and all their progress (chord scores) should be saved | User plays the game, then closes the app  This is the userdata file following this:    This shows that the code successfully saves user progress .  The app is then run again, and closed (Cmaj and Emaj were seen)  The csv file looks like this:    This is expected because the difference in the two files is +1 on Emaj and +1 on Cmaj | User plays the game again with the same chord PASS -packs and the scores are the same as what they were before the app closed |

All the tests have now been passed, and the score has been implemented, as well as loading/saving of user data.

### Listener Second Iteration

To gain more clarity on what needs to be fixed, I used my old project version, that had a graph in it of the FFT data. I played the chord whilst screen recording the result, then stepped through frame by frame to find the graph of the power of frequencies when the chord is first played.

First, I tested all the white notes on their own:

(Each note I played in the 4th then 5th octave, and the graphs are in that order)

|  |  |
| --- | --- |
| **Note** | **Graph** |
| C |  |
| D |  |
| E |  |
| F |  |
| G |  |
| A |  |
| B |  |

Something that I have noticed about these graphs is they start with more amplitude and get smaller and smaller as the frequency of the note increases. Furthermore, for each note there is one frequency which is higher than the rest of the frequencies heard, but the note being played does make more than one frequency. Higher notes only tended to make 2 or 1 frequencies whilst the lower notes had many.

Next, I did the same exercise but with the major chords:

|  |  |
| --- | --- |
| Chord | Graph |
| Cmaj  (C, E, G) |  |
| C#maj (C#, F, G#) |  |
| Dmaj  (D, F#, A) |  |
| Emaj |  |
| Fmaj |  |
| Gmaj |  |
| Amaj |  |
| Bmaj |  |

These chords follow a similar trend to the notes, where they decrease in amplitude as the frequency increases. They also all have many spikes.

I then decided to overlay the graphs on top of each other. This is Cmaj, with a C, E and G on top, respectively:

A graph with numbers and lines

Description automatically generated

A graph with numbers and lines

Description automatically generated

A graph with numbers and lines

Description automatically generated

There is definite correlation between the notes and the chord.

From these tests, I have gathered that:

* When a note is played, it is made up of multiple frequencies.
  + Verified by some quick research:
  + “A single piano note is made up of different related frequencies all occurring at the same time, many of these being ‘harmonics’ which we’ll describe in more detail in the next section. If you hit three, five or eight piano notes at the same time, then you are generating perhaps hundreds of frequencies all at once”.
    - <https://www.idrumtune.com/ultimate-guide-to-musical-frequencies/#:~:text=A%20single%20piano%20note%20is,of%20frequencies%20all%20at%20once>.
    - <https://en.wikipedia.org/wiki/Harmonic_series_(music)>

I found 3 methods which could help me: MFCC, HPS, and hamming/hanning windows. The hamming/hanning windows allow me to get rid of excess noise especially around the edges of the frequency spectrum and is the easiest to implement. The MFCC is a complicated algorithm that somehow allows for more accuracy and is used regularly with a Mel scale to determine the timbre of music. It is also used in speech recognition for more accurate sounds. An HPS (harmonic product spectrum) basically iterates through the frequencies and finds the product of the frequency and its harmonics, in place of that base frequency.

From all this research I’m going to first try some easier things to help it then add more advanced things when they don’t work. Firstly, a hanning window, which should help to reduce the huge spike at the start of the graphs.

The graphs now look like this:

A graph showing a blue line

Description automatically generated

Which is a bit neater, with no spikes at either end, but doesn’t solve the problem.

I also implemented an HPS. I used this equation:

A black rectangular object with a number and a square in the middle

Description automatically generated with medium confidence

Which iterates through the array finding the product of each note’s possible harmonics.

<http://musicweb.ucsd.edu/~trsmyth/analysis/Harmonic_Product_Spectrum.html>

A screenshot of a computer program

Description automatically generated

These two methods are my implementation of the HPS function. I normalised it like a standard normal distribution so that I could add a minimum threshold slider that changes how loud the noise must be in order for it to be counted as a note.

A screenshot of a computer

Description automatically generated

However, I ran into a lot of bugs, and I couldn’t see any improvement upon the actual note readings, so I reverted to the previous iteration of the listener module, but I kept the Hann window because that did help to smooth out the notes a bit. I also kept the slider for sensitivity.

One solution to the problem is arpeggiating the chords, so the listener only must worry about one note at a time. I will use this solution temporarily for the prototype so I can give the user something that works. (To change to arpeggiate mode, I just have the threshold return the highest value)

### Listener Second Iteration Tests

|  |  |  |  |
| --- | --- | --- | --- |
| F1.3a | Note is played and the note name is displayed | when A screenshot of a phone  Description automatically generated was played | PASS – the test is passed because an “A” note is dispayed |
| R1.3b | Note is played and the note name is displayed | when A group of colorful rectangular objects with black text  Description automatically generated with medium confidence  was played | PASS – the test is passed because an “A” note is dispayed |
| F1.4a | Multiple notes are played separately, and the program can list them | when  Pressed “A” the “C”  A piano keys with letters and numbers  Description automatically generated | PASS – the test is passed because an “A” note is dispayed |
| F1.4b | Multiple notes are played together, and the program can list them | A black background with white letters  Description automatically generated Was outputted when A C and E were pressed at the same time | PASS 50% - Although it did display the correct notes, because the notes are made up of lots of smaller notes, there were some extra notes played |
| R1.4c | Multiple notes are played together but they are the same | A black background with white letters  Description automatically generated Was outputted when A was pressed 3 times | PASS 50% - It correctly filtered out the repeated note but also thought the note A was made up of 3 different notes. |
| U1.5a | Background noise is played in the background | Was outputted when a and C where played at the same time as background noise | PASS – This test passes because no extra notes were outputted |
| R1.5b | Loud background noise is played in the background | A black background with white letters  Description automatically generated Was outputted When drilling noise was played at same time as ntoes | FAIL – because the notes were not the ones expected and there were additional notes, it fails this test. |
| R1.5c | Notes played that are far out of tune should be counted as background noise | A group of colorful squares with black text  Description automatically generatedWas played and nothing was outputted | PASS – the notes in the background were mostly ignored |

Predictably, tests failed that required more than one note to be played at any one time.

### Game Second Iteration

In this iteration, the user feedback requests that there be more emphasis on the streak and a more transparent performance. My plan to do this is to tell the user there in the middle of two chords streak when it is a multiple of five. I also want to have the colour of the chord dynamically change from green to red based on the score.



I added this method to expand on the sleeping on new question, which checks the score and displays it if needed.

I also updated the font size so that it would fit it: A computer screen with text

Description automatically generated

To have the score colour change dynamically:

A screen shot of a computer code

Description automatically generated

I get a value for red from the chords score.

There are no tests that failed for this module in the first iteration.

### UI - Third Iteration

This section I mostly will focus on completing the user feedback. This should include adding prompts to help the user get started, making the writing larger, removing the double click for selection of chords, and adding more emphasis on time to help increase stress.

**Prompts to help the user learn how to play the game:**

First, I added another label above the error label to assist the user. This meant I had to make the buttons span two rows, and so because the picture box I was using for the favourite button couldn’t span, I had to change the way that worked:



A black background with white text

Description automatically generated

This now works: A close-up of a computer screen

Description automatically generated

Now, I need to add the helpful messages:







A screen shot of a computer

Description automatically generated  
**Making the writing larger:**

I also changed some fonts to add clarity.

A screenshot of a computer

Description automatically generated

**Remove the double click to select:**

****

**More Emphasis on Time:**

For this one, I made the timer more noticeable, made it change colour and grow under 5 seconds.

A screenshot of a computer

Description automatically generated

I removed the help button, linking to tests 4.8 and 4.9 because of lack of time to implement, and given that it is a non-essential feature, it’s easy to get rid of. In the future, there should be a way to show the notes if the user gets it wrong.

I also temporarily swapped out the not-yet-implemented musical button for a high contrast button so my user that has dyslexia and can’t see well can still use this prototype.

A black background with white text

Description automatically generated

I made this button white text and black background and a font called Open-Dyslexic, which is meant to be easier to read.

When clicked, it turns the colour scheme to high contrast.

A black background with white text

Description automatically generated

## Prototype 2 Development Summary & Evaluation

At the end of this iteration, I tagged the commit “V2” on GitHub and this was the commit that my user’s reviewed.

### Testing Evaluation

For the Revision Engine module, all the previously failed tests have now passed. However, for the listener module, there are only slight improvements in the test results. The listener is slightly more accurate that the previous iteration, but the change is not big enough to pass all the tests. It accurately detects notes individually but not when more that one is played at any one time. I can tell the user to arpeggio the chords (play them one note at a time) and set the app to only detect one note per buffer, which would be a viable solution, except that full chords wouldn’t work.

## The Code

A screenshot of a computer

Description automatically generated

### ChordoMain.cs

**using** Chordo**.**Properties**;**

**using** NAudio**.**Wave**;**

**using** System**.**Windows**.**Forms**;**

**namespace** Chordo

**{**

**public** **partial** class ChordoMain **:** Form

**{**

RevisionEngine Rev**;**

Listener mic**;**

int time**;**

double Sensitivity**;**

/// <summary>

/// If True, Start when btn pressed, if false, stop.

/// </summary>

bool btnStartMode **=** **true;**

int streak **=** 0**;**

List**<**int**>** checkedPacks **=** **new** List**<**int**>();**

**public** static int MAXTIMEALLOWED **=** 15**;** //seconds

/// <summary>

/// gets the form going

/// </summary>

**public** ChordoMain**()**

**{**

// Start Form & Engines

InitializeComponent**();**

lblHelper**.**Text **=** "Hi! Please select at least one chord pack (on the left) to continue."**;**

Rev **=** **new** RevisionEngine**(**lblErrorOut**,** clbPacks**);**

mic **=** **new** Listener**();**

mic**.**minAmplitude **=** **(**double**)**sensitivitySlider**.**Value **/** 100**;**

// Reset to start

ResetAll**();**

**}**

/// <summary>

/// Resets the time, chord text, screen colour

/// </summary>

void ResetAll**()**

**{**

// Reset Heart to empty

btnHeart**.**BackgroundImage **=** Resources**.**emptyHeart\_removebg\_preview**;**

// reset time to max

time **=** MAXTIMEALLOWED**;**

lblTimer**.**Text **=** "\_\_"**;**

lblTimer**.**ForeColor **=** Color**.**Black**;**

// Reset visual feedback

NoScreen**();**

**}**

/// <summary>

/// On heart button clicked, favourite or unfavourite the chord

/// </summary>

/// <**param** name="sender"></**param**>

/// <**param** name="e"></**param**>

**private** void btnHeart\_Click**(object** sender**,** EventArgs e**)**

**{**

// if it's not favourited already, favourite and change heart colour

**if** **(**Rev**.**GetCurrentChord**().**favourite **==** **false)**

**{**

btnHeart**.**BackgroundImage **=** Resources**.**heart\_removebg\_preview**;**

lblHelper**.**Text **=** "Chord Favourited! You should see this chord more from now on."**;**

Rev**.**MakeFavourite**(true);**

**}**

// else unfavourite it and change to empty heart

**else**

**{**

btnHeart**.**BackgroundImage **=** Resources**.**emptyHeart\_removebg\_preview**;**

lblHelper**.**Text **=** "Chord Unfavourited!"**;**

Rev**.**MakeFavourite**(false);**

**}**

**}**

/// <summary>

/// Runs on start or stop button clicked, if start button then initilise a lot of the microphone and revision engine, and load the chord packs in the checked list.

/// If the stop button, then stop listening reset stuff.

/// </summary>

/// <**param** name="sender"></**param**>

/// <**param** name="e"></**param**>

**private** void btnStartStop\_Click**(object** sender**,** EventArgs e**)**

**{**

**if** **(**btnStartMode**)**

**{**

checkedPacks **=** **new** List**<**int**>** **{** **};**

//add each checked item to the list

**for** **(**int x **=** 0**;** x **<** clbPacks**.**Items**.**Count**;** x**++)**

**{**

**if** **(**clbPacks**.**GetItemChecked**(**x**))**

**{**

checkedPacks**.**Add**(**x**);**

**}**

**}**

//Shows error if no packs selected

**if** **(**checkedPacks**.**Count **==** 0**)**

**{**

lblErrorOut**.**Text **=** "You must select at least one chord pack to continue"**;**

**}**

**else**

**{**

lblHelper**.**Text **=** "Play the chord before the time runs out!"**;**

//Start the mic listening and reset

mic**.**StartListening**();**

NewQuestion**();**

btnStartStop**.**Text **=** "Exit"**;**

btnStartMode **=** **false;**

**}**

**}**

**else**

**{**

//disable timers

ListenTick**.**Enabled **=** **false;**

CountdownTimer**.**Enabled **=** **false;**

//stop listening

mic**.**StopListening**();**

ResetAll**();**

btnStartStop**.**Text **=** "Start"**;**

btnStartMode **=** **true;**

**}**

**}**

List**<**string**>** notesPlayed **=** **new** List**<**string**>();**

**public** int MAXNOTESBEFOREINCORRECT **=** 100**;**

// Pause length between questiosn

**public** int QUESTION\_TIMEOUT **=** 1000**;**

/// <summary>

/// runs at a regular interval based on the clock. analyses notes and checks whether they are in the chord or not.

/// </summary>

/// <**param** name="sender"></**param**>

/// <**param** name="e"></**param**>

**private** void ListenTick\_Tick**(object** sender**,** EventArgs e**)**

**{**

//Listen for new chord on tick

//Disable tick

ListenTick**.**Enabled **=** **false;**

//Listen

List**<**string**>** notesFound **=** mic**.**ProcessData**();**

//makes sure there are notes

**if** **(**notesFound **!=** **null** **&&** notesFound**.**Count **!=** 0**)**

**{**

//make sure none of the notes are repeated

**foreach** **(**string note **in** notesFound**)**

**{**

**if** **(!**notesPlayed**.**Contains**(**note**))**

**{**

notesPlayed**.**Add**(**note**);**

**}**

**}**

**foreach** **(**string note **in** notesPlayed**)** **{** Console**.**Write**(**note **+** ", "**);** **}**

Console**.**WriteLine**();**

//check if the notes are in the chord and do stuff based on that

bool correct **=** Rev**.**CheckNotes**(**notesPlayed**);**

**if** **(**correct**)**

**{**

CorrectScreen**();**

QuestionTimeOut**();**

NewQuestion**();**

**}**

**else** **if** **(**notesPlayed**.**Count **>** MAXNOTESBEFOREINCORRECT**)**

**{**

IncorrectScreen**();**

QuestionTimeOut**();**

NewQuestion**();**

**}**

**}**

//Enable tick

ListenTick**.**Start**();**

**}**

/// <summary>

/// Pause the system and streak displayed if multiple of 5

/// </summary>

**private** void QuestionTimeOut**()**

**{**

**if** **(**streak **%** 5 **==** 0 **&&** streak **!=** 0**)**

**{**

lblChord**.**Text **=** $"Your streak is {streak}!"**;**

lblChord**.**Font **=** **new** Font**(**"Segoe Print"**,** 24**,** FontStyle**.**Bold**);**

Application**.**DoEvents**();**

System**.**Threading**.**Thread**.**Sleep**(**QUESTION\_TIMEOUT**);**

lblChord**.**Font **=** **new** Font**(**"Segoe Print"**,** 72**,** FontStyle**.**Bold**);**

**}**

System**.**Threading**.**Thread**.**Sleep**(**QUESTION\_TIMEOUT**);**

**}**

/// <summary>

/// Gives a new chord

/// </summary>

**private** void NewQuestion**()**

**{**

ResetAll**();**

//Display Chord

lblChord**.**Text **=** Rev**.**NextChord**(**checkedPacks**).**name**.**ToString**();**

//set the colour of the chord to the colour relative to the score

int red **=** **(**int**)**Math**.**Round**(**255 **\*** **(**Rev**.**GetCurrentChord**().**score**));**

//Color color = Color.FromArgb(red, 128, 128);

//lblChord.ForeColor = color;

//Update the favourite icon accordingly

**if** **(**Rev**.**GetCurrentChord**().**IsFav**()** **==** **true)**

**{**

btnHeart**.**BackgroundImage **=** Resources**.**heart\_removebg\_preview**;**

**}**

//Start timer countdown from MAXTIMEALLOWED

CountdownTimer**.**Enabled **=** **true;**

notesPlayed **=** **new** List**<**string**>();**

ListenTick**.**Enabled **=** **true;**

int x **=** 2**;**

Console**.**WriteLine**(**$"Chord Name: {Rev.GetCurrentChord().name}\nFavourite: {Rev.GetCurrentChord().IsFav()}\nScore: {Rev.GetCurrentChord().score}\nNotes: {Rev.GetCurrentChord().GetNotesAsString()}\nTimes Played: {Rev.GetCurrentChord().timesPlayed}"**);**

**}**

/// <summary>

/// Displays the correct screen

/// </summary>

**private** void CorrectScreen**()**

**{**

Rev**.**CalcChordScore**(**MAXTIMEALLOWED **-** time**);**

lblChord**.**BackgroundImage **=** **new** Bitmap**(**Chordo**.**Properties**.**Resources**.**GreenTick**);**

lblChord**.**BackgroundImageLayout **=** ImageLayout**.**Zoom**;**

streak**++;**

lblStreak**.**Text **=** streak**.**ToString**();**

Application**.**DoEvents**();**

**}**

/// <summary>

/// Displays the incorrect screen

/// </summary>

**private** void IncorrectScreen**()**

**{**

Rev**.**CalcChordScore**(**MAXTIMEALLOWED**);**

lblChord**.**BackgroundImage **=** **new** Bitmap**(**Chordo**.**Properties**.**Resources**.**RedCross**);**

lblChord**.**BackgroundImageLayout **=** ImageLayout**.**Zoom**;**

streak **=** 0**;**

lblStreak**.**Text **=** streak**.**ToString**();**

Application**.**DoEvents**();**

**}**

/// <summary>

/// Undisplays the tick/cross

/// </summary>

**private** void NoScreen**()**

**{**

lblChord**.**BackgroundImage **=** **null;**

lblChord**.**BackgroundImageLayout **=** ImageLayout**.**Zoom**;**

Application**.**DoEvents**();**

**}**

/// <summary>

/// Runs every time the countdown timer ticks

/// </summary>

/// <**param** name="sender"></**param**>

/// <**param** name="e"></**param**>

**private** void CountdownTimer\_Tick**(object** sender**,** EventArgs e**)**

**{**

time**--;**

// check if time has run out

**if** **(**time **<** 0**)**

**{**

IncorrectScreen**();**

QuestionTimeOut**();**

NewQuestion**();**

**}**

// check if time is almost out

**if** **(**time **<** 5**)**

**{**

lblTimer**.**ForeColor **=** Color**.**Red**;**

lblTimer**.**Font **=** **new** Font**(**"Russo One"**,** 72**,** FontStyle**.**Underline**);**

**}**

lblTimer**.**Text **=** time**.**ToString**();**

**}**

/// <summary>

/// Runs on press of the skip button

/// </summary>

/// <**param** name="sender"></**param**>

/// <**param** name="e"></**param**>

**private** void btnSkip\_Click**(object** sender**,** EventArgs e**)**

**{**

IncorrectScreen**();**

QuestionTimeOut**();**

NewQuestion**();**

**}**

/// <summary>

/// Runs on index changed of pack selector

/// </summary>

/// <**param** name="sender"></**param**>

/// <**param** name="e"></**param**>

**private** void clbPacks\_SelectedIndexChanged**(object** sender**,** EventArgs e**)**

**{**

// update help messages

**if** **(**btnStartMode**)**

**{**

lblHelper**.**Text **=** "Great! You've selected a chord pack! If you're happy with your selection, get your piano ready and press 'Start'!"**;**

**}**

**else**

**{**

lblHelper**.**Text **=** "To change the chords you're playing, press stop, then select them."**;**

**}**

**}**

/// <summary>

/// on change of sens slider, change mic amp mins

/// </summary>

/// <**param** name="sender"></**param**>

/// <**param** name="e"></**param**>

**private** void sensitivitySlider\_Scroll**(object** sender**,** EventArgs e**)**

**{**

mic**.**minAmplitude **=** **(**double**)**sensitivitySlider**.**Value **/** 100**;**

**}**

/// <summary>

/// Toggle high contrast button

/// </summary>

/// <**param** name="sender"></**param**>

/// <**param** name="e"></**param**>

**private** void btnToggleHighContrast\_Click**(object** sender**,** EventArgs e**)**

**{**

**if** **(**lblChord**.**BackColor **==** SystemColors**.**Control**)**

**{**

lblChord**.**BackColor **=** SystemColors**.**ControlText**;**

lblChord**.**ForeColor **=** SystemColors**.**Control**;**

**}**

**else**

**{**

lblChord**.**BackColor **=** SystemColors**.**Control**;**

lblChord**.**ForeColor **=** SystemColors**.**ControlText**;**

**}**

**}**

**}**

**}**

### ChordoMain.Designer.cs

A screenshot of a computer

Description automatically generated

### ChordoMain.resx

A screenshot of a computer

Description automatically generated

### Chord.cs

**using** Accord**;**

**using** System**.**Web**;**

**namespace** Chordo

**{**

/// <summary>

/// Chord Object

/// </summary>

**internal** class Chord

**{**

List**<**string**>** notes**;**

**public** double score**;**

**public** string name**;**

**public** bool favourite**;**

**public** int time **=** ChordoMain**.**MAXTIMEALLOWED**;**

**public** int timesPlayed **=** 0**;**

/// <summary>

/// Construct the chord

/// </summary>

/// <**param** name="pName"></**param**>

/// <**param** name="numNotes"></**param**>

/// <**param** name="note1"></**param**>

/// <**param** name="note2"></**param**>

/// <**param** name="note3"></**param**>

/// <**param** name="note4"></**param**>

/// <**param** name="note5"></**param**>

**public** Chord**(**string pName**,** int numNotes**,** string note1 **=** **null,** string note2 **=** **null,** string note3 **=** **null,** string note4 **=** **null,** string note5 **=** **null)**

**{**

notes **=** **new** List**<**string**>();**

AddNote**(**note1**);**

AddNote**(**note2**);**

AddNote**(**note3**);**

AddNote**(**note4**);**

AddNote**(**note5**);**

name **=** pName**;**

score **=** 0**;**

**}**

**public** Chord**()**

**{**

**}**

/// <summary>

/// Adds a note to the chord

/// </summary>

/// <**param** name="note"></**param**>

**internal** void AddNote**(**string note**)**

**{**

**if** **(**note **!=** **null)**

**{**

notes**.**Add**(**note**);**

**}**

**}**

/// <summary>

/// Returns all the notes in the chord

/// </summary>

/// <**returns**>List<String> notes</**returns**>

**internal** List**<**string**>** GetNotes**()**

**{**

**return** notes**;**

**}**

/// <summary>

/// Returns the notes in the chord as a string

/// </summary>

/// <**returns**></**returns**>

**internal** string GetNotesAsString**()**

**{**

string output **=** ""**;**

**foreach** **(**string note **in** notes**)**

**{**

output **+=** note **+** ", "**;**

**}**

**return** output**;**

**}**

/// <summary>

/// Returns true/false dependant on the chord's favourite state

/// </summary>

/// <**returns**></**returns**>

**internal** bool IsFav**()**

**{**

**return** favourite**;**

**}**

**}**

**}**

### ChordPack.cs

**namespace** Chordo

**{**

/// <summary>

/// A collection of chords

/// </summary>

**internal** class ChordPack

**{**

List**<**Chord**>** chords**;**

string name**;**

/// <summary>

/// Constructor for if the chords in the pack are known

/// </summary>

/// <**param** name="pName"></**param**>

/// <**param** name="pChords"></**param**>

**public** ChordPack**(**string pName**,** List**<**Chord**>** pChords**)**

**{**

name **=** pName**;**

chords **=** pChords**;**

**}**

/// <summary>

/// Constructor for if the chords aren't known

/// </summary>

/// <**param** name="pName"></**param**>

**public** ChordPack**(**string pName**)**

**{**

name **=** pName**;**

chords **=** **new** List**<**Chord**>();**

**}**

/// <summary>

/// Adds a chord to the pack

/// </summary>

/// <**param** name="chord"></**param**>

**internal** void AddChord**(**Chord chord**)**

**{**

chords**.**Add**(**chord**);**

**}**

/// <summary>

/// Returns the list of chord in the pack

/// </summary>

/// <**returns**></**returns**>

**public** List**<**Chord**>** GetChords**()**

**{**

**return** chords**;**

**}**

/// <summary>

/// ToString override prints name and chord count

/// </summary>

/// <**returns**></**returns**>

**public** **override** string ToString**()**

**{**

string send **=** $"Chord Pack, '{name}', has {chords.Count} Chords in it"**;**

**return** send**;**

**}**

**}**

**}**

### ChordMap.cs

// Start of adapted code from https://joshclose.github.io/CsvHelper/examples/configuration/class-maps/mapping-properties/

**using** CsvHelper**.**Configuration**;**

**namespace** Chordo

**{**

**sealed** class ChordMap **:** ClassMap**<**Chord**>**

**{**

/// <summary>

/// Constructor maps vars to map

/// </summary>

**public** ChordMap**()**

**{**

Map**(**m **=>** m**.**name**);**

Map**(**m **=>** m**.**score**);**

Map**(**m **=>** m**.**favourite**);**

Map**(**m **=>** m**.**time**);**

Map**(**m **=>** m**.**timesPlayed**);**

**}**

**}**

**}**

// End of adapted code

### RevisionEngine.cs

**using** System**.**Text**.**RegularExpressions**;**

**using** System**.**Web**;**

**using** static System**.**Windows**.**Forms**.**VisualStyles**.**VisualStyleElement**.**TextBox**;**

**using** CsvHelper**;**

**using** System**.**Globalization**;**

**namespace** Chordo

**{**

/// <summary>

/// The class that handles learning and chord output

/// </summary>

**internal** class RevisionEngine

**{**

// Numbers between 0 and 1 that describes how much the different variables affects the chord's score

double timeEffect **=** 0.5**;**

double prevTimeEffect **=** 0.5**;**

double favouriteEffect **=** 0.5**;**

List**<**ChordPack**>** packs **=** **new** List**<**ChordPack**>();**

Chord currentChord**;**

**public** List**<**Chord**>** AllChords**;**

Label ErrorOut**;**

/// <summary>

/// Constructor, loads chords from file

/// </summary>

/// <**param** name="pErrorOut">Where errors should be outputted</**param**>

/// <**param** name="clbPacks">The listbox of packs</**param**>

**public** RevisionEngine**(**Label pErrorOut**,** ListBox clbPacks **)**

**{**

**this.**ErrorOut **=** pErrorOut**;**

AllChords **=** **new** List**<**Chord**>();**

**foreach** **(**string file **in** Directory**.**EnumerateFiles**(**@"..\..\..\..\Packs"**))**

**{**

//validation, don't add if pack hasn't got chords in it

ChordPack**?** chordpck **=** **(**loadChords**(**$"{file}"**));**

**if** **(**chordpck **!=** **null)**

**{**

clbPacks**.**Items**.**Add**(**File**.**ReadLines**(**file**).**First**());**

packs**.**Add**(**chordpck**);**

**}**

**}**

**for** **(**int i **=** 0**;** i **<** packs**.**Count**;** i**++)**

**{**

AllChords**.**AddRange**(**packs**[**i**].**GetChords**());**

**}**

LoadUserData**(**@"..\..\..\..\UserData.csv"**);**

**}**

Regex ScanChordsReg **=** **new** Regex**(**"(.+), (\\d), (.+), (.+), (.+);"**);**

/// <summary>

/// Loads all the chords in a pack from a text file, using REGEX. Text File in format "name, numNotes, note1{, note2, note3, note4, note5}\n"

/// </summary>

/// <**param** name="packAddress"></**param**>

/// <**returns**>a chord pack</**returns**>

/// <**exception** cref="NotImplementedException"></**exception**>

**private** ChordPack**?** loadChords**(**string packAddress**)**

**{**

// Read Text from file

string packText **=** File**.**ReadAllText**(**packAddress**);**

// match the regex

MatchCollection matches **=** ScanChordsReg**.**Matches**(**packText**);**

// display error if no matches

**if** **(**matches**.**Count **==** 0**)**

**{**

ErrorOut**.**Text **=** $"The chordpack {packAddress} didn't contain valid data or was empty."**;**

**return** **null;**

**}**

// make new pack where name is the first line

ChordPack pack **=** **new(**File**.**ReadLines**(**packAddress**).**First**());**

// for each chord, match

**foreach** **(**Match match **in** matches**)**

**{**

// match will be in format {CDName} {CDNumNotes} {CDNote 1} {Note 2} .. { Note x}

string CDname **=** match**.**Groups**[**1**].**Value**;**

int CDnumNotes **=** int**.**Parse**(**match**.**Groups**[**2**].**Value**);**

string CDnote1 **=** match**.**Groups**[**3**].**Value**;**

// Instantiate the chord

Chord CD **=** **new** Chord**(**CDname**,** CDnumNotes**,** CDnote1**);**

// add each note

**for** **(**int i **=** 1**;** i **<** CDnumNotes**;** i**++)**

**{**

CD**.**AddNote**(**match**.**Groups**[**i **+** 3**].**Value**);**

**}**

// Add the chord to the pack

pack**.**AddChord**(**CD**);**

**}**

**return** pack**;**

**}**

/// <summary>

/// Saves the Chord data to a csv file

/// </summary>

/// <**param** name="address"></**param**>

**private** void SaveUserData**(**string address**)**

**{**

//new csv file

**using** **(**var writer **=** **new** StreamWriter**(**address**))**

**{**

**using** **(**var csv **=** **new** CsvWriter**(**writer**,** CultureInfo**.**InvariantCulture**))**

**{**

// Write the chords to the csv file according to the map

csv**.**Context**.**RegisterClassMap**<**ChordMap**>();**

csv**.**WriteRecords**(**AllChords**);**

**}**

**}**

**}**

/// <summary>

/// loads the user data from address

/// </summary>

/// <**param** name="address"></**param**>

**private** void LoadUserData**(**string address**)**

**{**

// Check if the file exists

**if(**File**.**Exists**(**address**))**

**{**

// Open csv

**using** **(**var reader **=** **new** StreamReader**(**address**))**

**using** **(**var csv **=** **new** CsvReader**(**reader**,** CultureInfo**.**InvariantCulture**))**

**{**

csv**.**Context**.**RegisterClassMap**<**ChordMap**>();**

// Get chords

var records **=** csv**.**GetRecords**<**Chord**>();**

// Iterate over chords

**foreach(**var record **in** records**)**

**{**

// Give current chords new values

**foreach** **(**var OldChord **in** FindChords**(**record**.**name**))**

**{**

OldChord**.**score **=** record**.**score**;**

OldChord**.**favourite **=** record**.**favourite**;**

OldChord**.**time **=** record**.**time**;**

OldChord**.**timesPlayed **=** record**.**timesPlayed**;**

OldChord**.**name **=** record**.**name**;**

**}**

**}**

**}**

**}**

**}**

/// <summary>

/// Returns all the chords in AllChords with a specific name

/// </summary>

/// <**param** name="name"></**param**>

/// <**returns**>List of Chords</**returns**>

**private** List**<**Chord**>** FindChords**(**string name**)**

**{**

List**<**Chord**>** returnValues **=** **new** List**<**Chord**>();**

// Linear Search

**foreach(**var chord **in** AllChords**)**

**{**

**if(**chord**.**name **==** name**)**

**{**

returnValues**.**Add**(**chord**);**

**}**

**}**

**return** returnValues**;**

**}**

**private** Chord prevChord**;**

Random r **=** **new** Random**();**

/// <summary>

/// Chooses a next new chord, uses score

/// </summary>

/// <**param** name="chosenPacks"></**param**>

/// <**returns**></**returns**>

**public** Chord NextChord**(**List**<**int**>** chosenPacks**)**

**{**

// Make a list of possible chords

List**<**Chord**>** possibleChords **=** **new** List**<**Chord**>();**

**for** **(**int i **=** 0**;** i **<** packs**.**Count**;** i**++)**

**{**

**if** **(**chosenPacks**.**Contains**(**i**))**

**{**

possibleChords**.**AddRange**(**packs**[**i**].**GetChords**());**

**}**

**}**

// Remove previous chord from list

**if** **(**possibleChords**.**Contains**(**prevChord**))**

**{**

possibleChords**.**Remove**(**prevChord**);**

**}**

// Choose a chord

// Order the chords by descended score

possibleChords **=** possibleChords**.**OrderBy**(**a **=>** r**.**Next**()).**ToList**();**

possibleChords **=** possibleChords**.**OrderBy**(**x **=>** x**.**score**).**ToList**();**

possibleChords**.**Reverse**();**

int num**=**0**;**

// Choose the first, second or third chord

**if** **(!(**possibleChords**.**Count **<** 2**))**

**{**

num **=** r**.**Next**(**0**,** 2**);**

**}**

currentChord **=** possibleChords**[**num**];**

prevChord **=** currentChord**;**

// Save user data

SaveUserData**(**@"..\..\..\..\UserData.csv"**);**

**return** currentChord**;**

**}**

/// <summary>

/// Checks if the notes played are in the current chord

/// </summary>

/// <**param** name="notesPlayed">These are the notes that should be checked</**param**>

/// <**returns**></**returns**>

**internal** bool CheckNotes**(**List**<**string**>** notesPlayed**)**

**{**

// Linear Search

**foreach** **(**string note **in** currentChord**.**GetNotes**())**

**{**

// If not in notes

**if** **(!**notesPlayed**.**Contains**(**note**))**

**{**

**return** **false;**

**}**

**}**

**return** **true;**

**}**

/// <summary>

/// Calculates a new score for the chord

/// </summary>

/// <**param** name="time"></**param**>

**public** void CalcChordScore**(**int time**)**

**{**

// Keep track of timesPlayed

currentChord**.**timesPlayed**++;**

int favBoost **=** 0**;**

// Check if it's favourited or not

**if** **(**currentChord**.**favourite**)**

**{**

favBoost **=** 1**;**

**}**

// Calc score ( a number between 0 and 1)

double score **=** **(**timeEffect **\*** **(**time **/** 15**)** **+** prevTimeEffect **\*** **(**currentChord**.**time **/** 15**)** **+** favouriteEffect **\*** **(**favBoost**))** **/** currentChord**.**timesPlayed**;**

// DEBUGGING:

Console**.**WriteLine**(**$"SCORE CALC: new TIME: {time}, Prev TIME: {currentChord.time}, favBoost: {favBoost}, times Played: {currentChord.timesPlayed}, score: {score}"**);**

// Store score and time

currentChord**.**score **=** score**;**

currentChord**.**time **=** time**;**

**}**

/// <summary>

/// Gets the current chord

/// </summary>

/// <**returns**>currentChord</**returns**>

**internal** Chord GetCurrentChord**()**

**{**

**return** currentChord**;**

**}**

/// <summary>

/// makes the current chord favourited

/// </summary>

/// <**param** name="v"></**param**>

**internal** void MakeFavourite**(**bool v**)**

**{**

currentChord**.**favourite **=** v**;**

**}**

**}**

**}**

### Listener

**using** Accord**.**Math**;**

**using** Accord**.**Statistics**;**

**using** CsvHelper**.**Configuration**.**Attributes**;**

**using** Microsoft**.**VisualBasic**.**Logging**;**

**using** NAudio**.**Wave**;**

**namespace** Chordo

**{**

**public** class Listener

**{**

//Declare classes and variables

**public** WaveIn waveIn**;**

**public** BufferedWaveProvider bwp**;**

**private** int RATE**;**

**private** int BUFFERSIZE**;**

**public** int DEVICENUMBER**;**

**private** double threshold**;**

double**[]** hannWindow**;**

**public** double minAmplitude**;**

/// <summary>

/// Consructor - instantiates helper classes with values given, starts buffer.

/// </summary>

/// <**param** name="pBUFFERSIZE">Size of the buffer each sample | default: 8192 bytes</**param**>

/// <**param** name="pRATE">sample rate of the wave | default = 44100Hz</**param**>

/// <**param** name="pDeviceNumber">The device number of the microphone | default = 1</**param**>

**public** Listener**(**int pBUFFERSIZE **=** 8192**,** int pRATE **=** 44100**,** int pDeviceNumber **=** 1**,** double minAmplitude **=** 0**)**

**{**

RATE **=** pRATE**;**

BUFFERSIZE **=** pBUFFERSIZE**;**

// Initialise WaveIn class

waveIn **=** **new** WaveIn

**{**

DeviceNumber **=** DEVICENUMBER**,**

WaveFormat **=** **new** WaveFormat**(**RATE**,** 1**)**

**};**

bwp **=** **new** BufferedWaveProvider**(**waveIn**.**WaveFormat**);**

**this.**minAmplitude **=** minAmplitude**;**

**}**

/// <summary>

/// Event handler that activates when there is data available in the class.

/// </summary>

/// <**param** name="sender"></**param**>

/// <**param** name="e"></**param**>

**private** void waveIn\_DataAvailable**(object?** sender**,** WaveInEventArgs e**)**

**{**

//Add data to buffer

bwp**.**AddSamples**(**e**.**Buffer**,** 0**,** e**.**BytesRecorded**);**

**}**

/// <summary>

/// Starts listening to the mic

/// </summary>

**public** void StartListening**()**

**{**

// Starts the buffer

waveIn**.**DataAvailable **+=** **new** EventHandler**<**WaveInEventArgs**>(**waveIn\_DataAvailable**);**

bwp**.**BufferLength **=** BUFFERSIZE **\*** 2**;**

bwp**.**DiscardOnBufferOverflow **=** **true;**

// Starts recording the wave

waveIn**.**StartRecording**();**

**}**

/// <summary>

/// Stops listening to the mic

/// </summary>

**public** void StopListening**()**

**{**

waveIn**.**StopRecording**();**

**}**

/// <summary>

/// finds what note is a frequency by using midi nums. Contains validation to discard notes if they are too far away from the actual frequency.

/// </summary>

/// <**param** name="frequency"></**param**>

/// <**returns**></**returns**>

**public** string WhatNoteAmI**(**double frequency**)**

**{**

// Calculate MIDInum

double MIDInum **=** 12 **\*** Math**.**Log2**((**double**)**frequency **/** **(**double**)**440**)** **+** 69**;**

// Validation: too far away from actual frequency

**if(**Math**.**Abs**(**MIDInum **-** Math**.**Round**(**MIDInum**))>(**0.3**))**

**{**

**return** **null;**

**}**

// Round the MIDInum

int MIDInumRounded **=** **(**int**)**Math**.**Round**(**MIDInum**);**

string**[]** notes **=** **{** "A"**,** "A♯"**,** "B"**,** "C"**,** "C♯"**,** "D"**,** "D♯"**,** "E"**,** "F"**,** "F♯"**,** "G"**,** "G♯" **};**

// find the note

string note **=** notes**[((**MIDInumRounded **-** 21**)** **%** 12**)];**

**return** note**;**

**}**

/// <summary>

/// Process the data on tick

/// </summary>

/// <**returns**></**returns**>

**public** List**<**string**>** ProcessData**()**

**{**

//read bytes from bwp into frames

int frameSize **=** BUFFERSIZE**;**

var frames **=** **new** byte**[**frameSize**];**

bwp**.**Read**(**frames**,** 0**,** frameSize**);**

//check that the buffer isn't empty

**if** **(**frames**.**Length **==** 0**)** **return** **null;**

//if (frames[frameSize - 2] == 0) return null;

//pull PCM values from the buffer

// incoming data is 16-bit (2 bytes per audio point)

int BYTES\_PER\_POINT **=** 2**;**

// create a (32-bit) int array ready to fill with the 16-bit data

int PointCount **=** frames**.**Length **/** BYTES\_PER\_POINT**;**

// Generate the Hanning Window

GenerateHannWindow**();**

// Get the pcm values from the wave

double**[]** pcm **=** CalculatePCMValues**(**frames**,** PointCount**);**

// Apply the Hann Window

**for** **(**int i **=** 0**;** i **<** hannWindow**.**Length**;** i**++)**

**{**

pcm**[**i**]** **\*=** hannWindow**[**i**];**

**}**

// Get the fftReal Values

double**[]** fftReal **=** CalculateFFTRealValues**(**frames**,** PointCount**,** pcm**);**

// Adjust for Db

double**[]** fftDb **=** DbScale**(**fftReal**);**

// Normalise 0-1

fftDb**.**Normalize**(true);**

//double[] hps = CalculateHPS(fftReal);

//double[] NormalisedHPS = hps.Normalize();

// Return the notes played

**return** PullNotes**(**fftDb**,** fftDb**,** PointCount**);**

**}**

/// <summary>

/// Normalise

/// </summary>

/// <**param** name="hps"></**param**>

/// <**returns**></**returns**>

**private** double**[]** Normalise**(**double**[]** hps**)**

**{**

double**[]** Nhps **=** **new** double**[**hps**.**Length**];**

//using z-score normalisation

//calculate mean and standard deviation of the hps

double meanHps **=** hps**.**Mean**();**

double stdHps **=** hps**.**StandardDeviation**();**

//foreach element, center the values around the mean with a std dev of 1

**for(**int i **=** 0**;**i **<** Nhps**.**Length**;**i**++)**

**{**

Nhps**[**i**]** **=** **(**hps**[**i**]-**meanHps**)** **/** stdHps**;**

**}**

**return** Nhps**;**

**}**

/// <summary>

/// Calculates The HPS values for a given fft

/// </summary>

/// <**param** name="fftReal"></**param**>

/// <**returns**></**returns**>

**private** double**[]** CalculateHPS**(**double**[]** fftReal**)**

**{**

//Iterate through the positive half of the fft array

double**[]** hps **=** **new** double**[**fftReal**.**Length**];**

**for** **(**int i **=** 1**;** i **<** hps**.**Length**;** i**++)**

**{**

**if** **(**fftReal**[**i**]<**minAmplitude**)**

**{**

fftReal**[**i**]** **=** 0**;**

**}**

// Iterate through its harmonics (j = i, j = 2 \* i, j = 3 \* i, ...)

**for** **(**int j **=** i**;** j **<** fftReal**.**Length**;** j **+=** i**)**

**{**

// Multiply magnitudes of frequency and its harmonic:

**{**

hps**[**i**]** **+=** fftReal**[**i**]** **\*** fftReal**[**j**];**

fftReal**[**j**]** **=** 0**;**

**}**

**}**

**}**

**return** hps**;**

**}**

/// <summary>

/// Calculates the pcm values from the byte frames

/// </summary>

/// <**param** name="frames"></**param**>

/// <**param** name="PointCount"></**param**>

/// <**returns**></**returns**>

**public** double**[]** CalculatePCMValues**(**byte**[]** frames**,** int PointCount**)**

**{**

double**[]** pcm **=** **new** double**[**PointCount**];**

**for** **(**int i **=** 0**;** i **<** PointCount**;** i**++)**

**{**

// read the int16 from the two bytes

var val **=** BitConverter**.**ToInt16**(**frames**,** i **\*** 2**);**

// store the value

pcm**[**i**]** **=** **(**double**)(**val**);**

**}**

**return** pcm**;**

**}**

//Some code borrowed from ScottPlot

/// <summary>

/// Generates a Hann window in global space

/// </summary>

**private** void GenerateHannWindow**()**

**{**

hannWindow **=** **new** double**[**BUFFERSIZE **/** 2**];**

var angleUnit **=** 2 **\*** Math**.**PI **/** **(**hannWindow**.**Length **-** 1**);**

**for** **(**int i **=** 0**;** i **<** hannWindow**.**Length**;** i**++)**

**{**

hannWindow**[**i**]** **=** 0.5 **\*** **(**1 **-** Math**.**Cos**(**i **\*** angleUnit**));**

**}**

**}**

//End of code borrowed from ScottPlot

/// <summary>

/// Calculates real fft values

/// </summary>

/// <**param** name="frames"></**param**>

/// <**param** name="PointCount"></**param**>

/// <**param** name="pcm"></**param**>

/// <**returns**></**returns**>

**public** double**[]** CalculateFFTRealValues**(**byte**[]** frames**,** int PointCount**,** double**[]** pcm**)**

**{**

double**[]** fftReal **=** **new** double**[**PointCount **/** 2**];**

//get fft values

var fft **=** FFT**(**pcm**);**

// Copy only the first half because the second half is imaginary

Array**.**Copy**(**fft**,** fftReal**,** fftReal**.**Length**);**

**return** fftReal**;**

**}**

/// <summary>

/// Applies the FFT

/// </summary>

/// <**param** name="data"></**param**>

/// <**returns**>Array of double[] where the first half is the imaginary parts</**returns**>

**public** double**[]** FFT**(**double**[]** data**)**

**{**

double**[]** fft **=** **new** double**[**data**.**Length**];**

System**.**Numerics**.**Complex**[]** fftComplex **=** **new** System**.**Numerics**.**Complex**[**data**.**Length**];**

// Populate the array with complex numbers where the real part is the data

**for** **(**int i **=** 0**;** i **<** data**.**Length**;** i**++)**

fftComplex**[**i**]** **=** **new** System**.**Numerics**.**Complex**(**data**[**i**],** 0.0**);**

// Apply the transform

Accord**.**Math**.**FourierTransform**.**FFT**(**fftComplex**,** Accord**.**Math**.**FourierTransform**.**Direction**.**Forward**);**

// Take the magnitude of the complex data

**for** **(**int i **=** 0**;** i **<** data**.**Length**;** i**++)**

fft**[**i**]** **=** fftComplex**[**i**].**Magnitude**;**

**return** fft**;**

**}**

/// <summary>

/// Adjusts the amplitudes of the fft values for the Db scale

/// </summary>

/// <**param** name="fftReal"></**param**>

/// <**returns**></**returns**>

**public** double**[]** DbScale**(**double**[]** fftReal**)**

**{**

double**[]** fftRealDB **=** **new** double**[**fftReal**.**Length**];**

int a **=** 0**;**

// Iterate over the array and apply a (log base 10 )\*10

**foreach** **(**var f **in** fftReal**)**

**{**

var y **=** **(**Math**.**Log10**(**f**)** **\*** 10**);**

fftRealDB**[**a**]** **=** y**;**

a**++;**

**}**

**return** fftRealDB**;**

**}**

/// <summary>

/// Get the notes from an array of fft values, including validation: variable threshold,

/// </summary>

/// <**returns**></**returns**>

**public** List**<**string**>** PullNotes**(**double**[]** hps**,** double**[]** fftIn**,** int PointCount**)**

**{**

int PrevI **=** 0**;**

List**<**string**>** notes **=** **new** List**<**string**>();**

// Create a threshold based on hps/amplitudes

threshold **=** Calibrate**(**hps**);**

// Iterate over the array

**for** **(**int i **=** 0**;** i **<** fftIn**.**Length**;** i**++)**

**{**

// If the amplitude is within the threshold and far enough above 0 and far enough away from the prev datum

**if** **(**hps**[**i**]** **>=** threshold **&&** i **>** 10 **&&** i**-**PrevI**>**5**)**

**{**

// Find the frequency for the datum

double frequency **=** **(**i **\*** RATE**)** **/** PointCount**;**

/\* DEBUG CONSOLE OUT

if (frequency != 0)

{

Console.WriteLine("Freq:" + frequency + " " + i + " " + fftIn[i] + " " + threshold + " " + minAmplitude);

}\*/

//

// Find the not name related to the freq (returns null if unvalid)

string note **=** WhatNoteAmI**(**frequency**);**

// Make sure the freq was valid

**if** **(**note **!=** **null)**

**{**

// Add note if note already in list

**if** **(!**notes**.**Contains**(**note**))**

**{**

notes**.**Add**(**note**);**

**}**

**}**

PrevI **=** i**;**

**}**

**}**

**return** notes**;**

**}**

/// <summary>

/// Make a new threshold from the amplitudes of the fft/hps

/// </summary>

/// <**param** name="fftIn"></**param**>

/// <**returns**></**returns**>

**private** double Calibrate**(**double**[]** fftIn**)**

**{**

double highest**=**0**;**

double secondHighest **=** 0**;**

double thirdHighest **=** 0**;**

double fourthHighest **=** 0**;**

// find highest, second highest, third ...

**for** **(**int i **=** 0**;** i **<** fftIn**.**Length**;** i**++)**

**{**

**if** **(**fftIn**[**i**]** **>** highest**)**

**{**

fourthHighest **=** thirdHighest**;**

thirdHighest **=** secondHighest**;**

secondHighest **=** highest**;**

highest **=** fftIn**[**i**];**

**}**

**else** **if** **(**fftIn**[**i**]** **>** secondHighest**)**

**{**

fourthHighest **=** thirdHighest**;**

thirdHighest **=** secondHighest**;**

secondHighest **=** fftIn**[**i**];**

**}**

**else** **if** **(**fftIn**[**i**]** **>** highest**)**

**{**

fourthHighest **=** thirdHighest**;**

thirdHighest **=** fftIn**[**i**];**

**}**

**if** **(**fftIn**[**i**]** **>** highest**)**

**{**

fourthHighest **=** fftIn**[**i**];**

**}**

**}**

//Console.WriteLine(fourthHighest + " " + thirdHighest + " "+ secondHighest + " "+ highest);

// NOTE: if only one note is to be heard, only need to return highest

/\*

if (fourthHighest > minAmplitude)

{

return fourthHighest;

}

else if(thirdHighest > minAmplitude)

{

return thirdHighest;

}

else if(secondHighest > minAmplitude)

{

return secondHighest;

}

else\*/

**if** **(**highest **>** minAmplitude**)**

**{**

**return** highest**;**

**}**

**else** **{** **return** 1000000**;** **}**

**}**

**}**

**}**

### Program.cs

**namespace** Chordo

**{**

**internal** static class Program

**{**

/// <summary>

/// The main entry point for the application.

/// </summary>

**[**STAThread**]**

static void Main**()**

**{**

// To customize application configuration such as set high DPI settings or default font,

// see https://aka.ms/applicationconfiguration.

ApplicationConfiguration**.**Initialize**();**

Application**.**Run**(new** ChordoMain**());**

**}**

**}**

**}**

## Defining Post-Development Black Box Tests

F = Functionality

U = Usability

R = Robustness

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Black Box Test | Feature | Description | Test Data/Process | Expected Result | Justification |
| F1a | 1.1, 1.2, 1.3 | The app lists the note that was played | “C” note played during the game phase | The error/info box shows “C” | This test proves that the app can detect and identify a note being played through the mic |
| F1b | 1.1, 1.2, 1.3 | The app lists the note that was played | “E” note played during the game phase | The error/info box shows “E” | This tests if other notes work than C, to be thorough. |
| F1c | 1.1, 1.2, 1.3 | The app lists the note that was played | “A” note played during the game phase | The error/info box shows “A” | This tests if other notes work than C, to be thorough. |
| F1d | 1.4 | The notes are saved, and duplicate notes are not |  |  | This tests if duplicate notes are counted twice so the app doesn’t deduct points |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

# Evaluation

## Post-development testing

I am going to do black box testing here that was described at the end of the post dev stage. The test data is similar but not the same to the white box testing in the development stage.

A close-up of black text

Description automatically generated

## Testing Evaluation



## Unmet Success Criteria

A close up of black text

Description automatically generated

## Usability Evaluation



## Future Development of Unmet Success Criteria



## Maintenance

Maintenance - Funding – open source? Closed source? Subscription?



## Future Improvements



TODO:  
USER FEEDBACK  
Change Success Criteria Wording THROUGHOUT

REFORMAT ALGORITHMS

EXPAND REV ENGINE

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)
4. Interaction Design Foundation - IxDF. (2016, August 31). What is Visual Hierarchy? Interaction Design Foundation - IxDF. https://www.interaction-design.org/literature/topics/visual-hierarchy [↑](#footnote-ref-4)