University of Nottingham

MUSI 3071

Team 4

Melodify - An Emotive Decision Tree

Authors
Jonathan Moore, Christopher Wainwright, James Bateman, Benjamin Goringe,
Cheuk Yin Tze, Amelia Walastyan, Ngoc Nguyen

January 6, 2025

Contents

1 Introduction

Melodify is an application designed to enhance midi instrument practice and improvisation through the guidance of a midi-based decision tree. The project aims to transform these activities through 'guided improvisation' where a player is given the opportunity and encouraged to experiment with alternative harmonies and melodies as they perform a piece. Conceptually, the project can be described as a "choose your own adventure"-type interface, where notes played on the input instrument influence the direction of musical development.

When learning music traditionally, novice musicians face arduous practice sessions, which can be repetitive and uninspiring, leading many to give up early before they get a chance to discover the true creativity of musicianship. This application aspires to alleviate this issue. First of all, the application provides a temporal, piano-roll interface which is much more intuitive to a new learner. Second, the application takes performers on a journey of discovery, suggesting alternative melodies that they could try and eliminating the static, repetitive nature associated with a traditional practice approach.

The application achieves this through a combination of internal pre-composed midi sections, a provided backing track, and the pygame library's midi input functionality. As the player plays, notes from the current section flow from the top to the bottom of the screen, and playing a note with the correct tone and timing destroys the note. The gamified piano roll described seamlessly blends between multiple midi sections (indicated by different note colours). When the note of a new section is played, the game switches to the target section and continues to progress as normal.

Melodify is targeted at beginner musicians looking to start learning an instrument or exploring the musical world, and more experienced players who are looking for a new musical experience with their instrument.

2 User Experience Journey

2.1 User Experience Flow

Melodify is designed to provide users with the most realistic and fun experience to learn piano quickly. That is why it receives inputs from a real keyboard, which is connected to the laptop externally, through a USB or MIDI cable. The app supports various kinds of keyboards, ranging from large e-pianos to small MIDI pianos.

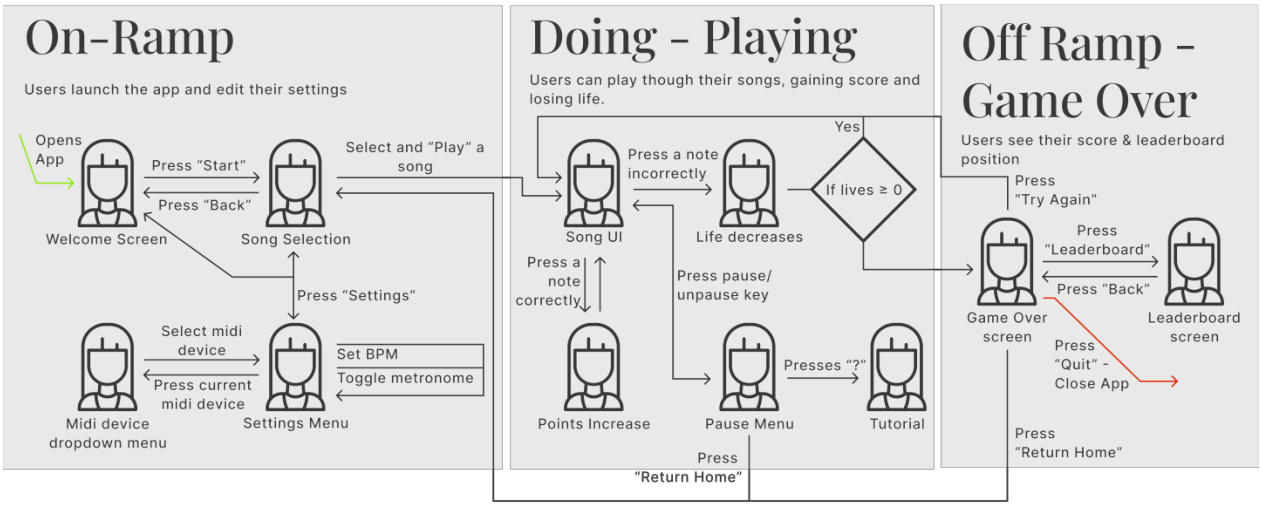


Figure 1: Flowchart of User Experience Journey

Thanks to the intuitive design of the interface, the user experience is straightforward. After plugging the keyboard into a computer, users open Melodify app. The application automatically detects the keyboard and connects it. If many keyboards are connected, users can select the desired one on "Setting" page by pressing "Settings" button. To begin the learning journey, users press the "Start" button, then choose a song to learn and press the "Play" button to start the game.

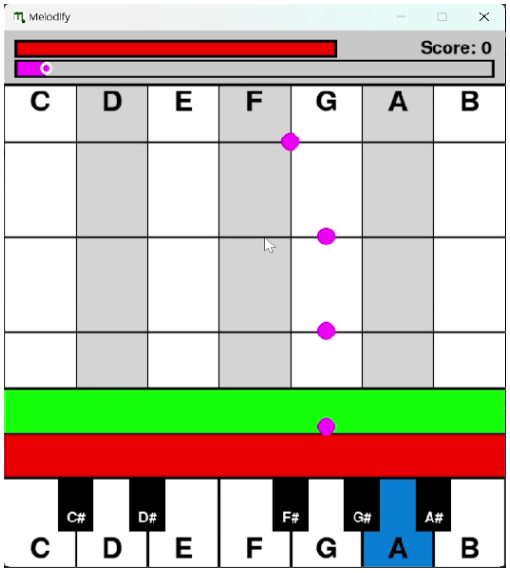


Figure 2: Notes falling from the top of screen

Once the game begins, the notes (dots) are displayed and they fall from the top down vertically. There is a virtual keyboard layout at the bottom of the app, which represents the keys on the user's keyboard. Whenever

users press any keys on their physical keyboards, the corresponding keys on the virtual keyboard in the app will be shaded, giving users an intuitive experience. Therefore, users can adapt quickly and self-correct easily when pressing the wrong keys.

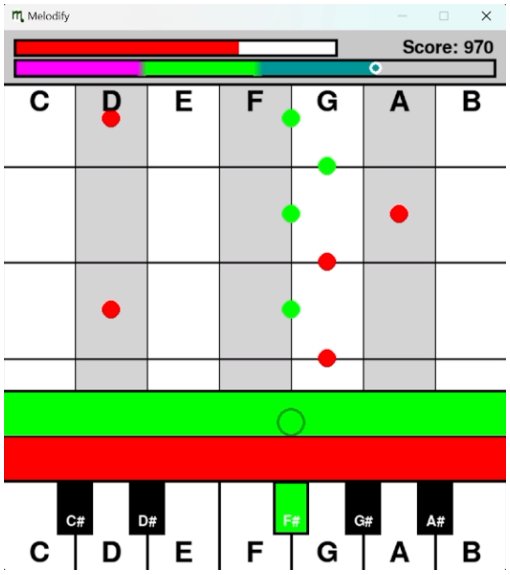
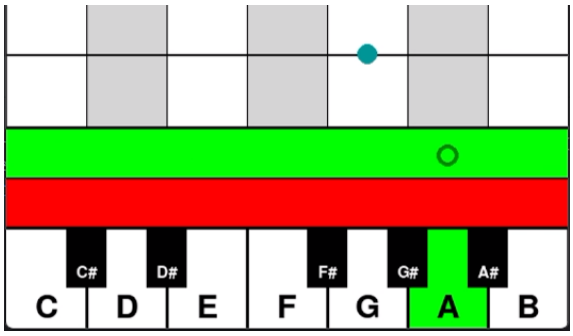
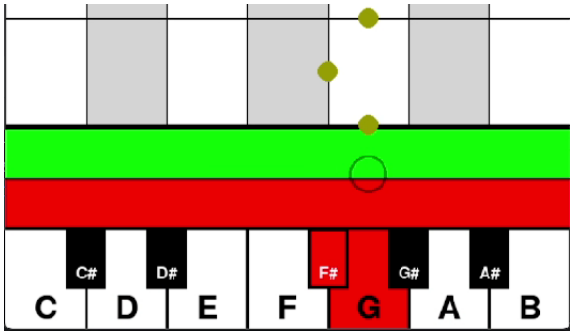


Figure 3: Users choose either of 2 branches

Sometimes the application displays 2 series of notes (2 branches) with 2 different colours, users need to choose a path by pressing a note associated with their preferred branch. The falling notes gradually go down until they reach the green bar, right above the virtual keyboard, which indicates the perfect time for users to press the corresponding keys.



(a) User presses the correct key at right time



(b) User presses the wrong keys

Figure 4: The virtual keyboard turns green for correct keys and red for wrong ones.

Whenever the users press the correct keys at the right time, the virtual keyboard turns green to celebrate that success, and the score is increased. On the other hand, when they press the wrong keys, or press at the wrong time, the virtual keyboard turns red and the health bar (life) is deducted.

During the game, users can press “P” key to pause the game. On the Pause Menu page, users can access the game tutorial by pressing “?” key, and pressing it again to exit the tutorial.

As the players progress, the game gradually enhances the difficulty level. More notes of the song are displayed, which makes the users play quicker, therefore, the piano sound becomes more beautiful and complex. As the game progresses, the system plays more well-composed background music tracks. That means the longer the players play, the more beautiful music they produce. Therefore, this mechanism

makes the game sticky to the players, and the more they practice playing piano, the better they become.

When users make mistakes, such as pressing the wrong keys or pressing the correct key at the wrong time, the health bar is deducted. The game ends when the health bar is empty, and then users go to the "Game Over" page. On this page, users can check their rankings by clicking "Leaderboard" button, play again by pressing the "Play Again" button or they can stop the game by choosing the "Quit" button.

After experiencing the game, we want the users to feel playing piano is not as difficult as they thought, it's not boring and time-consuming like what people say. They can play their favourite songs and enjoy beautiful rhythms on the first several tries. The feeling our product brings to users is the love of playing the piano. Therefore, they want to use Melodify to practice the skill and see it as a trusty tool to guide them in learning piano.

2.2 How To Play - Summary Of User Experience Flow

1. Connect the Keyboard: Plug your keyboard into the computer using a USB or MIDI cable.
2. Press "Start" button in the menu to begin. Choose your favourite song and press "Play".
3. Press the corresponding keys to falling notes on the screen, on your keyboard.
4. Press the correct keys at the right time to turn the keyboard green. Missed notes or wrong timing will turn your virtual keyboard red and reduce your health bar.
5. Pause the game by pressing "P" key and access this tutorial by pressing the "?" key on your keyboard.
6. When two branches of notes with 2 different colours appear, choose a path by pressing a note associated with your preferred branch.
7. The game ends when your health bar is empty. You can choose to play again or quit.

3 Design of the MMR Experience

3.1 Core Features

Our decision-based digital score allows users to affect a song's progression without having musical knowledge. Introducing branches into the music encourages people to make natural decisions about how the music should progress. This design decision is consistent with user control and freedom in Human-Computer Interaction principles, allowing users to customise their musical experience based on how they think the song should be narrated.

We used colour coding to distinguish different paths in the branching system, making it more intuitive. This design employs the Gestalt resemblance concept, allowing users to link each branch with its associated colour instantly. Colour difference decreases cognitive strain, allowing users to concentrate on the aural experience without being distracted too much by decision-making. Furthermore, this technique encourages playful, exploratory involvement, which increases engagement by letting users feel personally connected to the music.

The implementation of the Score and Health Bar in real time: The score and health bar are very useful in making the user engaged and providing him or her the feedback. These features are in consistent with the feedback and visibility paradigm in HCI. This makes the users know the consequences of their actions as they happen, whether positive or negative. The score makes the user want to hit the right notes and this increases when one is told to try again. The health meter adds an element of risk into the interaction in a very subtle manner and this includes the idea of difficulty. When the health of a user is depleted due to wrong hit of the notes, then the user has to start the song again which is a natural process that enhances the user's skills. This mechanic is in line with flow theory in UX design since it does not create a situation where the user is either faced with a very easy task that they can do with their eyes closed or a very difficult one that will make them want to throw the device around.

User-Friendly Interface. The interface design adheres to core HCI principles to provide an intuitive and engaging user experience. For example, when users press incorrrect notes, the wrong keys illuminate red, and correct notes turn green. This immediate visual feedback adheres to consistency and error prevention principles by reinforcing the user's knowledge of their performance using familiar red-for-error, green-for-success cues. This dynamic feedback method reduces ambiguity, allowing users to self-correct and adapt quickly. Furthermore, the design adheres to the aesthetic-usability effect; making the interface visually appealing and easy to navigate improves perceived usability and encourages further contact. Through these thoughtfully created features, we ensure that the system captivates users and adheres to proven HCI principles and UX/UI best practices.

Colour Reflecting on Mood of Song. An essential feature of our idea is introducing colour to reflect how the branch the user takes affects the song. This makes the user experience more intuitive and engaging because assigning colour gives the user better context, which allows for less second-guessing and paints a better vision of what the user wants the song to sound like. For example, shades of blues convey calm and serene melodies and Warmer colours, like shades of orange, will hint more towards lively, upbeat rhythms. We leverage the psychology of colour to communicate colour to the mood of the music and also assign the Gestault principle of resemblance to create seamless associations.

3.2 User Scenarios

Emma loved to play the piano but never got to a point of being good enough to create her own music. Then she came across Melodify software that promised she could play all the songs that she wanted, while the AI made her change the music in ways she wanted. As such, she had to try it out and see what it could offer her, such a promise When the music begins, she sees two options which are different in color; blue and yellow. She wanted to know what kind of impact her decision would have and hence she selected the blue one. She notices this affected the song in a way that made the melody softer and more gentle. In the middle of the song, she is also presented with an option to choose between orange and blue. Out of sheer

interest to see what impact orange would have on the song, she selected it. This transformed the song into a more energetic and fast paced one and she loved how this altered the story of the song and therefore, she continued picking the branches which were coloured red or orange.

A College student, Jonny wants to use our platform to enhance his timing and rhythm skills. He decides to pick the song "Jingle Bells", and it can be seen that there is a score and a health bar which resets depending on the player's timing and accuracy. The first time he tried it, he was not very good at keeping time and having good accuracy and he did not even get to the end of the song before he lost. He felt quite annoyed and then have a go at it again to see if he could finish the song. After some time he was able to complete the song. He felt proud but saw a low score in his achievement. This made him continue training and to work on his accuracy and time. Though the practice sessions were quite fascinating, he had to work hard and get a perfect score.

3.3 Interactivity

The interactivity is the heart of our experience, making the user to be able to control the song narrative. Our product gives people the ability to control the narrative of any song and this creates a journey that is dynamic and engaging. Rather than simply being able to read, and play music and have it be something that is simply listened to, users are able to compose their own music. Being able to control the music as you go can make for a very engaging experience and provide a new form of expression for many beginner music lovers.

When discussing how to enhance the experience through group meetings and user feedback, the confusion that people had with the branches was removed and colour to represent the emotions was added. This helped in making the users understand the concept better and ensured that they were not clicking on anything and everywhere with no understanding of what would happen. The problem was solved and this in turn helped to enhance the users' emotional connection with the music.

The use of audio layering has also improved this connection where the layers of music also help in enhancing the feelings that the music is attempting to bring out thus having a greater influence on the decisions that the user makes. The audio layering is also vital in helping the users on the timing of the songs they are trying to play since it is their ears that they use in determining the right time.

The users are able to make decisions and modify the songs in accordance with their ideas, as they progress through the users' journey, it is unique. We think that this idea can provide a unique level of personal creativity and makes every song to be the representation of user's art. This changes the role of the listener in music as well as the ability to build and rearrange the musical award that is presented to them.

4 Role of Music and Audio in the Experience

Music and audio were at the heart of our project, serving as both the medium and the message in delivering an immersive, interactive experience. By leveraging adaptive audio design, creative composition, and real-time feedback mechanisms, we enabled users to engage deeply with their musical creativity while reinforcing the goals of interactivity and engagement. This section will explore the intricate design and implementation of music and audio within the project, highlighting the adaptive design, compositional choices, challenges, and areas for future improvement.

4.1 Adaptive Music Design: Personalised Musical Narratives

The foundation of our project lay in adaptive music design, where user decisions dynamically shaped the musical progression. We utilised a decision-tree framework to structure the musical flow, allowing users to create personalised narratives. This system presented players with choices at specific decision points, and their selections determined the melody, harmony, or rhythm of the subsequent sections. This interactivity empowered users to feel as though they were composing their own unique musical stories, ensuring their engagement and satisfaction.

For instance, when a user selects a particular melodic pathway, the system adapts the harmony and instrumentation to align with their decision. These changes were subtle yet impactful, maintaining cohesion within the overall composition while reflecting the user's input. This adaptability created a deeply personalised experience, where no two playthroughs were the same.

The integration of a MIDI keyboard enhanced this adaptive framework. By capturing user input in real-time, the MIDI system allowed for immediate auditory feedback, ensuring seamless transitions and dynamic responses to user actions. The decision to use MIDI technology was pivotal, as it bridged the gap between digital interactivity and traditional music performance, making the experience feel authentic and intuitive. The real-time responsiveness of MIDI input eliminated delays that could disrupt the flow, ensuring users remained immersed in their musical exploration.

We based the entire demonstration on the song Faded by Alan Walker, specifically leveraging its iconic chord progression vi-IV-I-V (in the key of G major). From this foundation, we composed 14 original variations inspired by the motif of Faded. Each variation introduced unique elements, such as changes in pitches and rhythm, while maintaining the emotional resonance of the original motif. These variations and the backing tracks were composed in Logic Pro X, enabling high production quality and seamless integration into the prototype. This approach demonstrated the system's ability to adapt to user decisions while offering a dynamic musical experience.

4.2 Creative Decisions in Composition: Balancing Variety and Cohesion

The composition process played an equally critical role in achieving the project's goals. Our team focused on creating music that struck a balance between variety and cohesion. While each decision point introduced a new musical direction, we carefully crafted underlying themes and motifs to ensure the experience felt unified. This approach maintained an overarching musical identity, even as users explored different pathways.

In Logic Pro X, we created modular backing tracks that could be layered or adjusted depending on user input. The workflow started with programming the basic chord progression (vi-IV-I-V) using MIDI tracks. From there, we added layers of virtual instruments, including pads for atmospheric depth, rhythmic elements for maintaining tempo, and melodic lines to highlight user decisions. Each instrument track was carefully automated to respond dynamically to transitions, ensuring smooth shifts as users progressed through the experience.

Across the 14 variations, we experimented with a wide range of musical elements to provide users with diverse yet cohesive options. For example, some variations emphasised ambient textures with slow-moving

harmonic changes, while others introduced upbeat rhythms and brighter instrumental timbres. These compositional choices ensured that users experienced a sense of variety while the central theme remained recognisable.

To achieve high-quality audio, we utilised Logic Pro X's suite of tools, such as the Drummer Track for realistic percussion and the Alchemy Synth for creating lush, evolving textures. Effects such as reverb, delay, and EQ adjustments were applied to specific elements to enhance the immersive quality of the tracks. This meticulous approach to composition and production helped us deliver an adaptive, polished musical experience.

One of the most significant compositional elements was the inclusion of these backing tracks, which served multiple purposes. They acted as both a rhythmic foundation and a means of enhancing the overall atmosphere. These tracks helped users maintain timing, providing an intuitive sense of when to hit notes. Additionally, they created the feeling of being part of a collaborative musical performance, which testers frequently highlighted as one of the project's most engaging aspects.

4.3 Real-Time Feedback: Encouraging Engagement and Improvement

Real-time feedback mechanisms were integral to the user experience, providing immediate responses to user actions and reinforcing a sense of accomplishment. Each successful note was accompanied by auditory and visual cues, such as a pleasant chime or a colour change on the screen, signalling to users that they were performing correctly. Conversely, missed notes triggered subtle yet distinct cues, such as a muted tone or a brief flash of red, encouraging users to adjust without breaking the immersive flow.

These feedback systems were designed to strike a balance between guidance and autonomy. By avoiding overly intrusive cues, we allowed users to focus on their musical expression while still providing the necessary support to improve their performance. This approach was particularly effective for beginner users, who reported feeling motivated to refine their skills after receiving clear and constructive feedback.

In addition to individual note feedback, the system also provided real-time updates on overall performance. For instance, the health bar replaced traditional scoring systems to give users a more intuitive sense of how well they were playing. Starting with a full health bar that decreased with missed notes, this system aligned more closely with user expectations and enhanced the clarity of their progress. Testers responded positively to this change, highlighting its simplicity and effectiveness.

4.4 Immersion Through Audio Design

Immersion was a key design goal, and audio played a central role in achieving it. From the choice of instruments to the spatial arrangement of sound, every element was carefully considered to draw users into the experience. For example, we experimented with directional audio to create a sense of depth and space, where certain notes or effects seemed to originate from different parts of the virtual environment. This subtle yet effective technique enhanced the realism of the experience, making users feel as though they were surrounded by the music they were creating.

Another crucial factor in immersion was the timing and synchronisation of audio elements. Early prototypes revealed challenges in this area, particularly when users struggled to match their actions to the musical cues. To address this, we introduced a metronome feature and refined the backing tracks to provide a clearer rhythmic guide. These additions significantly improved users' ability to stay in sync with the music, contributing to a more cohesive and engaging experience.

5 Summary of Technical Realisation

Our codebase is organised into multiple files, each fulfilling distinct single responsibilities. This is good because it lets us have modular, robust development, giving us the best development environment in a distributed team, where several of us are working on the codebase simultaneously (Harding 2015).

5.1 `main.py`

5.1.1 Purpose

The `main.py` file is the entry point for the program (Figure 5). It parses command-line arguments, sets up logging, and launches the primary game loop by instantiating and running an instance of the `Game` class.

5.1.2 Overview of Code

```
import argparse
import logging
from game.game import Game

def main() -> None:
    parser = argparse.ArgumentParser()
    parser.add_argument(
        "--log-level",
        default="WARNING",
        choices=["DEBUG", "INFO"],
        help="Set the logging level"
    )
    args = parser.parse_args()

    logging.basicConfig(
        level=args.log_level,
        format="%(asctime)s [%(levelname)s] %(name)s: %(message)s"
    )

    game = Game()
    game.run()

if __name__ == '__main__':
    main()
```

Figure 5: Main Function

- Command-Line Interface (CLI) & Logging Config
  - The script takes a `--log-level` argument. This is to allow the program to be run in "debug" and "release" modes. This improves the program as logging is not performant and the end user should not need to see any logs.
  - I have used python's built-in logging library as it is the easiest and most performant way of implementing different log levels.
- Game Instance Creation: A `Game` object (defined in 5.2 `game.py`) is instantiated and then launched with its `run()` method.

5.2 game.py

5.2.1 Role of the Game Class

The Game class is at the heart of Melodify. It coordinates game states, manages MIDI input, orchestrates audio playback, and performs the real-time logic needed to update and render the user interface.

5.2.2 Game States

To maintain a clear flow, the app utilises state machine, i.e. an enumerated set of states (GameState):

- HOME (showing the main menu)
- SONG_SELECTION (selecting songs to play)
- PLAYING (core gameplay)
- PAUSE (pausing the game)
- GAME_OVER (end-of-game screen)
- TUTORIAL, LEADERBOARD, SETTINGS (supplementary screens)

Switching states happens when the user takes actions (e.g. clicking "Start", pausing, running out of health). Using a state machine architecture is good as it isolates each state the app can be in, which avoids accidental collisions between different rendering or logic paths, as well as making the game easy to extend.

5.2.3 Main Game Loop

```
# A simplified outline of the contents of game.py

while self.running:
    events = pygame.event.get()
    for event in events:
        if event.type == pygame.QUIT:
            self.state = GameState.QUIT

    if self.state == GameState.HOME:
        # Draw home screen, handle clicks
    elif self.state == GameState.SONG_SELECTION:
        # Draw song selection screen
    elif self.state == GameState.PLAYING:
        # Update pressed keys, update logic, and draw the gameplay
    ...

    pygame.display.flip()
    self.clock.tick(60)

pygame.quit()
```

Figure 6: The Main Game Loop

1. Event Handling: The loop starts by collecting recent events from the pygame event queue, checking for window close requests, button clicks, and key presses.
2. State-Specific Logic: A bonus of using a state machine architecture, the logic is separated for each state, with each calling it's own drawing function (e.g., draw_home_screen) and input handler (e.g., handle_home_screen_click).

3. Frame Control: `self.clock.tick(60)` makes the game aim to run at 60 frames per second, offering consistent timing for note falling animations and input polling. This is not ideal as it depends on the users' hardware being good enough to run the game at 60fps otherwise animations and timings would desync from things like the backing track. However for a simple prototype it is sufficient.

5.2.4 MIDI Input Management

`pygame.midi` is used to detect keyboard presses (note-on events) and releases (note-off events), allowing the program to get data from things like physical keyboards (Figure 7). We chose to use `pygame.midi` because it provides a direct integration with the rest of the pygame library, reducing overhead and complexity that would come with needing to bridge an different library.

```
def update_pressed_keys(self) -> None:
  if not self.midiInput:
      return
  if not self.midiInput.poll():
      return

  for event in self.midiInput.read(10):
      data = event[0]
      if isinstance(data, list):
          status, note, velocity, _ = data
          tone = Tone.fromMidi(note)
          if status == 144:
              self.pressedKeys[tone] = velocity > 0
          elif status == 128:
              self.pressedKeys[tone] = False
```

Figure 7: Handling MIDI Events

1. Polling
   - `midiInput.poll()` checks if any events are waiting. If `True`, `read(10)` means we can get up to ten events, which makes sense as the average person has 10 fingers and so can play 10 notes simultaneously.
   - By polling and processing MIDI data in each frame, the app is made sure to be maximally responsive to the user's input.
2. Note-On / Note-Off: In the MIDI protocol, status=144 means a note-on event (with velocity). A velocity of zero in some devices also indicates a note-off, status=128 is a dedicated note-off message (Association 2022).
3. Tone Conversion: `Tone.fromMidi(note)` transforms the MIDI note integer (e.g., 60 for middle C) into the `Tone` enum.

5.2.5 Health, Score, and Note Logic

When a note passes through the "hit zone" and the user presses the correct key, the score is increased, with an animation played to tell the user where in the "hit zone" the key was pressed. Missing the note results in a health deduction. Once health reaches zero, the game transitions to `GAME_OVER`.

5.3 `subScreens.py`

5.3.1 Purpose and Content

All of the screens (Home, Pause, Song Selection, Leaderboard, Game Over, etc.) are written in `subScreens.py`. Each screen has its own draw function and click handler.

- `draw_home_screen()`: Renders the main menu with “Start”, “Tutorial”, “Settings”, and “Quit” buttons.
- `draw_song_selection_screen()`: Displays a list of songs to choose from.
- `draw_pause_screen()`: Applies a translucent overlay to create a blurred background for the pause menu.
- `draw_game_over_screen()`: Shows final score, “Play Again” option , ”Leaderboard” option and ”Quit” option.

5.3.2 Click Handlers

Each click handler (Figure 8) works in the same way:

```
# Example of a click handler
def handle_home_screen_click(events) -> str | None:
  mouse_pos = pygame.mouse.get_pos()
  for event in events:
      if event.type == pygame.MOUSEBUTTONDOWN and event.button == 1:
          if (button_x <= mouse_pos[0] <= button_x + button_width
                  and button_y <= mouse_pos[1] <= button_y + button_height):
              return "start"
          ...
      elif event.type == pygame.QUIT:
          return "quit"
  return None
```

Figure 8: The Click Handler

Each of the click handlers do two things:

- Mouse Click Detection: Positions are compared against the bounding box of buttons, and if the user clicks inside any button’s region, an action string is returned (e.g., `"start"`).
- Return Values: Instead of directly changing the game state, the function returns a string (or `None`). The main loop in `game.py` then uses this value to switch to the correct state or take an action.

Using this design pattern ensures each screen is self-contained: it only needs to compute user interactions and does not need to know about global state transitions, making the code cleaner and easier to maintain (Harding, 2015).

5.4 note_data.py

5.4.1 Managing Musical Branches and Notes

As discussed in Section 3.3, one of the main creative concepts of Melodify is the idea of "branching" within a piece of music, allowing a user to choose different melodic or harmonic paths. `note_data.py` supplies two key classes:

1. `NoteData`: Which holds the timing, duration, and pitch (`Tone`) of an individual note.
2. `Branch`: Which holds a set of notes loaded from a specific MIDI track and references the next potential branches (Figure 9). Representing branches as objects in an OOP style is good because it makes data easier to manage and extend (Harding, 2015).

```
class Branch:
    def loadMidi(self) -> Notes:
      ...
      for event in events:
          current_time += int(event["time"])
          current_tone = Tone.fromMidi(int(event["note"]))
          match (event["type"]):
              case "note_on":
                  if not on_off[current_tone]:
                      on_off[current_tone] = current_time
              case "note_off":
                  if start_time := on_off[current_tone]:
                      notes.append(NoteData(
                          time=start_time / MIDIBEATLENGTH,
                          duration=(current_time - start_time) / MIDIBEATLENGTH,
                          tone=current_tone,
                          branch=self,
                      ))
                      on_off[current_tone] = None
      ...
      return notes
```

Figure 9: Code to extract MIDI Data

- `MIDIBEATLENGTH`: Defines how many "ticks" in the MIDI file correspond to a single beat in the game. Changing this value is how the BPM setting functions to change the speed of everything.
- `current_tone`: Raw MIDI note values are converted to the `Tone` enum. This approach safeguards against typos and simplifies the game logic by registering multiple octaves of the same note to a single tone.
- `start_time`: Captured on `note_on`. The difference between `note_off` and the last `note_on` event for the same tone determines the note's duration.
- Addition of songs: Additional MIDI files can easily be added as the app automatically converts them into `NoteData` and `Branch` objects, which can be easily used to extend the possible branching paths.

5.5 player.py

5.5.1 Audio Playback

This module manages background music playback using the pygame.mixer library, providing additional real-time feedback for the user playing. The `Music` class (Figure 10) acts as a wrapper for the pygame music player. The class offers the following methods:

- `play()`: Starts playback from the beginning.
- `pause()`: Temporarily halts the audio.
- `stop()`: Completely ends the audio.
- `unpause()`: Resumes from paused state.

```
import pygame
from pathlib import Path

class Music:
  def __init__(self, file: Path):
      pygame.mixer.init()
      pygame.mixer.music.load(file)
      self.paused = False

  def play(self):
      pygame.mixer.music.play()

  def pause(self):
      pygame.mixer.music.pause()

  def stop(self):
      pygame.mixer.stop()

  def unpause(self):
      pygame.mixer.music.unpause()
```

Figure 10: The Music Player Class

We decided to use pygame.mixer because it is included within pygame and provides sufficient audio features for our basic playback requirements.

5.6 ui.py

5.6.1 Drawing the Piano Roll and Feedback

All of the visible elements, i.e. falling notes, the piano keyboard, health bar, score, and progress bars etc. are handled here. Some notable functions are:

- `drawPiano()`: Renders a seven-note-wide representation of the user's piano keys at the bottom of the screen, highlighting them in different colours based on user input and correctness.
- `drawNote()`: Positions each note on the screen according to its timing.
- `drawHealth()` and `drawScore()`: Renders the user's health bar and current score at the top.
- `beatsToY()`: Converts a note's "beat time" to a y-coordinate, enabling the "falling note" effect.
- `draw_ghosts()`: Draws a "expanding drop" circular indicator when the user correctly presses a note inside the correct time zone.

- `drawProgressBar()`: Draws the user's journey through the piece, with each coloured section representing the colour of the branch the user was in at that time.

By separating the logic (in `game.py`) from the drawing routines (in `ui.py`), the app maintains clean architecture by allowing each file to remain focused with single responsibility, making the code more maintainable (Harding, 2015).

5.7 `settings.py`

Using a consolidated place for all settings gives the benefit of isolating important variables in a single file, simplifying their use over multiple places and for testing or customisation by decreasing the use of "magic numbers". This guarantees consistent modifications throughout the entire app (Martin 2009). It includes adjustable variables like:

- `BPM`: Beats per minute used to convert time steps to beats, controlling note fall speed.
- `MAX_HEALTH`: Determines how many mistakes a player can make before the game ends.
- `NOTE_DISPLAY_HEIGHT` and `WIDTH_SCALE`: Control the layout of the falling note area and piano key widths.
- `MIDI`: A boolean flag that can toggle MIDI input on or off if required.
- Colour definitions: Specifying user interface colours (health bar, button states, text).

6 Conclusion

Melodify represents a significant step forward in the way musicians interact with a musical score, especially through learning and creating music. The integration of human-computer interaction principles, adaptive audio design and gamification help the project provide an engaging and immersive platform for both novice musicians learning how to play the piano for the first time, as well as more experienced musicians looking for a new way to explore their musical skills creatively. It achieves this by bridging the gap between traditional musical education and more modern, interactive technologies that prioritise user engagement and adaptability.

The user friendly design of Melodify's interface underlines its accessibility to a diverse audience, with features such as real time visual and audio feedback, and gamification techniques such as a score and health bar, empowering users to experiment with musical scores without requiring advanced musical knowledge. This aligns with the broader goal of simplifying the music learning process and fostering creativity through the exploration of musical decision trees with the inclusion of feedback mechanisms allowing users to receive constant guidance, promoting learning and improvement while maintaining a sense of enjoyment and individual learning.

Critical analysis of our project highlights areas for further improvements. The use of a musical decision tree offers a personalised experience, the reliance of pre-defined decisions limit the extent of user-driven creativity. The use of an advanced machine-learning algorithm would enable a significantly wider range of playable music more tailored to the musician, massively increasing the potential for creativity within our project. Along side this, expanding the library of songs would help cater the project to a wider audience than the prototype is able to, allowing users with different preferences (ie. jazz, blues, etc) to play the kind of music they want to, enhancing the project's inclusivity and broad appeal.

Further areas for improvement can be made through the addition of collaborative playing and composition, allowing multiple users to interact simultaneously, either locally or over the internet, facilitating social learning and community-building among musicians. This would allow Melodify to follow modern trends in online education, where collaborative tools are increasingly used and valued for their ability to enhance learning and foster a sense of community.

From a technical perspective, Melodify's reliance on MIDI as an input mechanism can be seen as limiting the accessibility of the project, forcing users to use compatible devices to access the program. This factor could be significantly reduced by allowing a wider range of input types, such as a microphone or DAW. However, this was unachievable for our project due to a lack of technical experience in this field and a lack of time and money for development.

The use of adaptive compositions, layered accompaniments and real-time user feedback mechanisms demonstrates a comprehensive understanding of the interplay between the musical score and the user experience. Allowing musicians to influence the harmonic, melodic and rhythmic aspects of the music helps foster a sense of ownership and artistic expression, not only enhancing engagement but also contributing to a stronger appreciation of musical structure and creativity.

Melodify successfully combines the musical score with digital technologies to create a platform that reinvents musical education and creativity using modern trends and techniques, addressing key challenges in traditional musical eduction, together with offering an interactive and fun experience. While there are opportunities for improvements due to limitations of the teams knowledge and scale, particularly in adaptability and accessibility, the minimum viable prototype successfully meets the goals we initially laid out at the beginning of our project. By inspiring both aspiring and advanced musicians to explore and learn, Melodify shows the potential to leave a lasting impact on the way people learn and experience the musical score.

7 Appendix

GitLab:
https://projects.cs.nott.ac.uk/psybg3/musi3071-emotive-musical-decision-tree

Capture of Prototype:
https://www.youtube.com/playlist?list=PL3NdP9cBhJqxt8QMijgVOGOK6N7Obp2YM

References

Association, MIDI (2022). *MIDI message Reference*. URL: ttps://midi.org/spec-detail.

Harding, C. (2015). *Integrated Design and Construction - Single Responsibility: A Code of Practice*. John Wiley & Sons.

Martin, R. C. (2009). *Clean code: a handbook of agile software craftsmanship*. Pearson Education.