Piano Master

Daksh Dhaker (2022CS51264)
Aviral Singh (2022CS11290)

Introduction

Game Cause

Today, the visually impaired community encounters significant challenges in accessing educational resources, particularly in learning musical instruments like the piano. Traditional methods of teaching rely heavily on visual cues, rendering them inaccessible to those with visual impairments.
This innovative game aims to bridge the gap by providing an inclusive and accessible platform for individuals with visual impairments to learn and master piano playing skills. The game offers a multi-sensory learning experience, accommodating various learning styles and preferences.

Integrating the cause with game

In our game, we've incorporated a narration feature to guide you through the menu system. As soon as you start the game, a friendly voice will walk you through everything you need to know. You'll hear instructions on gameplay, learn about different modes, and get tips on navigating the menu. Need to select a song or adjust the pitch? The narrator will explain how to do it step by step. We want to make sure every player feels comfortable and confident as they explore the game.
We present three distinctive modes within our game, each offering a unique and engaging experience.

- Firstly, the Single Player mode provides an opportunity for individuals to learn and practise their favourite songs and pitches on the piano at their own pace. This mode serves as a personalised learning environment where players can hone their skills and master their chosen tunes.
- Next, the Versus Mode introduces an exciting competitive element, pitting players against each other in a dynamic piano-playing showdown. Participants are challenged to perform well on randomly selected songs, with scoring based on their accuracy and proficiency. This mode fosters friendly competition and encourages players to showcase their musical abilities in a spirited atmosphere.
- In the third mode, named the Cooperative Mode, players are not partners but rather opponents engaging in a collaborative challenge. Here, participants exchange musical notes with their opponents, aiming to perform the assigned melodies with precision. Scoring is based on the accuracy and proficiency of each player in executing the given tunes. This mode fosters strategic thinking, communication, and adaptability as players navigate the dynamic exchange of musical compositions in a competitive yet cooperative setting.

Advantages

The advantages of this game are as follows:

- **Battle Mode for Tutorial Purposes:** The third mode, Battle Mode, doubles as a tutorial platform where players can teach each other tunes and note sequences. This fosters peer-to-peer learning, with the scoring system providing feedback on how accurately tunes are replicated.

- **Single Player Mode for Practice Pitch:** The Single Player mode offers a Practice Pitch feature, allowing players to train themselves before venturing into multiplayer modes. Random pitches are played, testing players' ability to perform on unfamiliar notes, enhancing their adaptability and skillset.

- **Accessibility for Blind and Sighted Players:** Designed primarily for blind individuals, the game's narration and guidance are delivered through sound. Navigation is facilitated using arrow keys, and an info present as the first option on each provides page descriptions. Moreover, the scoring system is conveyed audibly, ensuring accessibility for all players.

- **Compatibility with Various Piano Systems:** The game seamlessly integrates with both electrical and virtual piano systems, accommodating players with diverse setups. It can be connected to any electrical piano system or utilised with virtual piano software, ensuring accessibility and convenience for all players, regardless of their equipment.

Implementation Highlights

Menu System

The menu system's attributes are:

- **Menu Navigation with Arrow Keys:** The menu system allows users to navigate between options using arrow keys. When hovering over an option, its name is narrated to the user, providing auditory guidance.
- **Selection by Pressing Enter:** Users can select a particular game or option by pressing the Enter key. This streamlined selection process enhances usability and accessibility.
- **Distinct Menu Classes:** Each menu within the system is organised into its own class, inheriting from a parent class called Menu.h. This design choice improves code readability and maintainability, allowing for efficient instantiation and management of menu elements.
- **Designed Using Pygame:** The entire menu system is built using Pygame, a popular Python library for game development.
- **Special Menu Screens:** There are some menu screens which are different from others like the screen in single player mode when the game starts, the screen where users select the desired parameters, the screen of results and the game screen in multiplayer mode. But how to navigate through these screens, this information is given by the info option present as the first option on each page.

Tech Stack

- pygame
- mido
- midi
- azure
- socket

Implementation Highlights

Single Player Menu

This class inherits from the Menu class. The insights of single player menu are:

- **Single Player Game Options:** Upon entering the Single Player game, users are presented with three options: Learn Pitch, Learn Songs, and Practice Pitch.
- **Learn Pitch:** In this mode, players are exposed to different pitches played in sequence, enabling them to familiarise themselves with the positions of keys on the piano and their corresponding sounds.
- **Learn Songs:** Players can select the songs they wish to learn in this mode. This option provides a varied selection of tunes for players to practise and master.
- **Parameters page:** In each of the three options first players have to set the parameters like whether you want to hear the piano key names or key pitches, similarly they also have to set the note length before starting the game.
- **Results Page:** At the conclusion of each session, players are presented with their scores, indicating their performance during the game.

Implementation Highlights

The MIDI System

We have ensured our game is as universal as possible by working directly with the MIDI standard (available with almost any electric piano). We have used pygame.midi to interface with Microsoft's internal synthesiser called the GS Wavetable Synth, and play notes. We're using mido to parse MIDI files.

We have a narrator class that can call out note names, as well as play the actual notes, given a MIDI file.

Base Game

The Base Game (MidiGame) is the parent of our game modes.
On initialising it with either an array of midi events or a file, it instantiates a narrator which narrates the note names and can also play the actual notes of the file/array on a separate thread.

Scoring System

The scoring system allocates positive 100 points if you've played the correct note in a 2 second interval of when the note should've been played. This is very lenient as of now as the games are supposed to appeal to beginners. For every incorrect/unnecessary note you play 50 points are deducted.

Implementation Highlights

Custom Socket Protocol (over TCP)

Since we had to send large python data structures over sockets, we had to package them properly. All our socket communications have a header of 4 bytes, which contains the length of our message in bytes. While receiving we read the header, then read exactly that many bytes.

Increasing Accessibility (Server on Azure)

We have deployed our server on Azure, as we don't want a blind person to deal with typing in random IP Addresses. Our clients contain a static ip address that points to our server instance.
All you need to play a multiplayer game is an Internet Connection.

Listener System

The Listener System is used in Battle Mode. It transcribes in real time whatever you've played into a MIDI event array.

The Single Player Mode

In the single player mode, we have some simple songs added for demo. You have to play along with the narrator's voice or notes. In the end you will get a score that tells you how well you did.
We also have a Learn Pitch mode that teaches you two of the most common octaves on the piano. In the Pitch Training mode, we play a certain number of random notes and you have to play them back.

Implementation Highlights

Client-Server Network (Implementation details)

Here we just used socket, _thread, struct and pickle libraries and the other functions implemented either from scratch or using these libraries in this network.
In the game's networking system, the client-server interaction is managed through the implementation of a separate class called network.py. When a client initialises an instance of this class, it establishes a connection with the server and communicates via signals. Notably, a single server handles both Battle Mode and Versus Mode sessions.

Within network.py, functions such as send and recvall facilitate the transmission and retrieval of information to and from the server. The send function encodes data using pickle before transmission, while recvall utilises the struct library to handle large arrays of musical tunes effectively.

The server employs a function named threaded_client to manage client connections. Upon establishing a connection, the client is assigned to a specific game session based on its designated game mode. Subsequently, a new thread is initiated for each player, allowing for concurrent handling of multiple game sessions. Each thread modifies the game state based on signals received from the corresponding client. The updated game state is then shared with both clients to ensure synchronisation of gameplay. This approach enables players to stay informed about their opponent's actions and current game state.

In the event of a player disconnecting from the server, the corresponding game session is terminated, and any remaining players are redirected to the results page. This streamlined approach ensures efficient management of game sessions and seamless transitions for players throughout their gameplay experience

Implementation Highlights

How versus and battle mode work (Interaction through server)

In the multiplayer mode of the game, which includes both the Versus Mode and the Battle Mode, client-server interaction is essential for facilitating gameplay between multiple players. To establish this communication, the game utilises Python libraries such as socket, _thread, pickle, and struct.

When a player initiates the game and selects the Versus Mode, a network connection is established between the client and the server. The game mode parameter is set to 0 to indicate Versus Mode. The player enters a queue, awaiting an opponent. Once another player with the same game mode joins the queue, the server pairs them together, assigning different game IDs but placing them in the same game session. Throughout the game, players' scores are calculated locally on their respective devices. Upon completion of the game, each player's final score is transmitted to the server using socket functions. Before transmission, the score information is encoded and packaged using the pickle library. The server then relays this score data to the respective opponents, allowing each player to view their own score and their opponent's score.

In the Battle Mode, the game mode is set to 1, and players enter a waiting queue similar to the Versus Mode. Once two players are matched, the game begins, consisting of two rounds. In the first round, players take turns playing challenge tunes they wish to present to their opponent. These tunes are stored as arrays. When the round ends, the arrays are serialised using the pickle.dumps method and sent to the server. In the second round, referred to as the Listening Round, players receive their opponent's challenge tunes and play them accordingly. Scores are calculated based on the performance in both rounds, utilising the same scoring system as in Versus Mode. This process is repeated twice to allow players to gain a better understanding of their opponent's style and abilities. After the completion of the two rounds, the final scores are displayed, concluding the Battle Mode gameplay session.

Evaluation Metrics

The criterion on which, our game should be evaluated can include:

- **Effective Client-Server Communication** - As explained above, we have efficiently made a communication system between client-server capable of even transmitting large sound arrays also.
- **Punctual Transitioning and Narration**-All the screen transitions and narrations are quite decent in timing as well as they are responsive too.
- **Advanced Scoring Mechanism** - Explained earlier.
- **Compatibility with Various Piano Types** - As explained above, our game is compatible with any electrical and virtual pianos.
- **Distinct and efficient game logic for each mode and its control**- Each game mode has its distinct logic since each game-mode works differently.
- **Structured Object-Oriented Programming**- We have written the code in a way that each menu screen, game, network, server, and client has its own class so that code is well organised and structured.

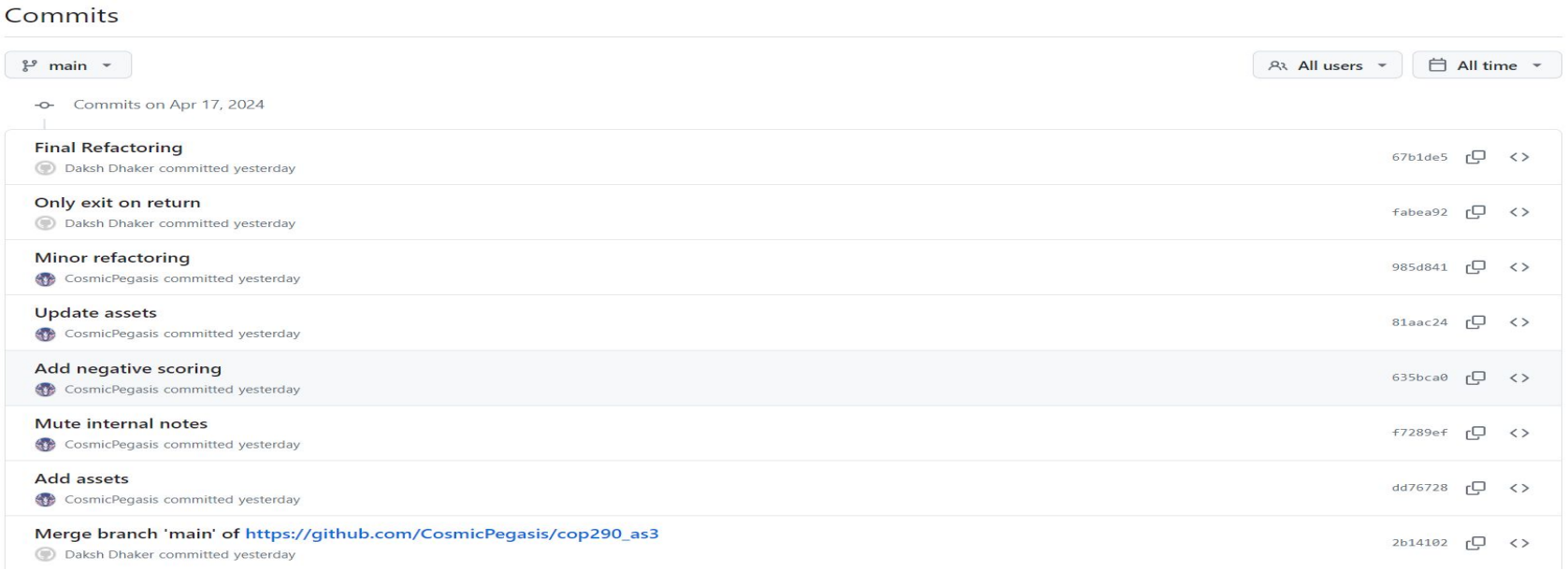
Snapshots

1. Home Page
   This is the home page, where most of the information about the game is narrated as the window opens, players can mute this info using arrow keys.

Snapshots

2. Main Menu (Only some of the snapshots are shown here to given an idea about the user interface)

Git Commits


Commits
main
All users
All time
Commits on Apr 17, 2024
Final Refactoring
Daksh Dhaker committed yesterday
67b1de5
Only exit on return
Daksh Dhaker committed yesterday
fabea92
Minor refactoring
CosmicPegasis committed yesterday
985d841
Update assets
CosmicPegasis committed yesterday
81aac24
Add negative scoring
CosmicPegasis committed yesterday
635bca0
Mute internal notes
CosmicPegasis committed yesterday
f7289ef
Add assets
CosmicPegasis committed yesterday
dd76728
Merge branch 'main' of https://github.com/CosmicPegasis/cop290_as3
Daksh Dhaker committed yesterday
2b14102

Merge branch 'main' of https://github.com/CosmicPegasis/cop290_as3
Daksh Dhaker committed yesterday
2b14102

Add assets
Daksh Dhaker committed yesterday
fb8d779

Fix disconnection client side logic
Daksh Dhaker committed yesterday
731333b

Fix Practice Game
Daksh Dhaker committed yesterday
a681b1c

Improve scoring algo
CosmicPegasis committed yesterday
fad7009

squash different socket handshake bug
CosmicPegasis committed yesterday
ea61444

fix practice mode
Daksh Dhaker committed yesterday
9e6428e

squash multiplayer bug
Daksh Dhaker committed 2 days ago
61bcf4d

Commits on Apr 16, 2024

Fix ip address
Daksh Dhaker committed 2 days ago
16b47c9

Reformat
CosmicPegasis committed 2 days ago
ca62014