Music Teacher C# outline

Overview:

This ear training music teacher app will help users learn music scales / chords / intervals while developing their ears to recognize the sound patterns involved in learning these concepts.

The piano: The training piano will have 3 octaves of notes that a user can manually play either by pressing the keys on the screen or pressing the keyboard button that corresponds with the note (one octave at a time, ex: ‘a’ corresponds with C2, ‘w’ corresponds with C#2: see reference at bottom of document). When a note is played the key should turn red.

Training mode: Buttons on the bottom of the screen will display the available scales / chords / intervals that a user can learn. When the user presses a button the corresponding scale / chord / interval will be played. The buttons on the piano should turn red when this happens and the finger positions will be displayed beneath the keyboard (optional).

Testing mode: In this mode the user will choose if they want to be tested on scales / chords / intervals. When they are ready they can press a button that will play a random scale / chord / interval and the user will have to select which scale / chord / interval they think was played.

User options:

* Toggle on / off visual piano: The piano visual on helps user learn the piano keys. Turning the piano visual off helps users learn by ear alone since they cannot use the visual to guide them.
* Toggle on / off piano labels: The labels on the keys can be turned off once a user is comfortable with the piano layout.
* Toggle on / off random mode: In simple mode all scales / chords / intervals will be played (trained / tested) around middle C. Every pattern will start on middle C, this makes learning the initial scales / chords / intervals easy for beginners. Advanced users can toggle on random mode where all scales / chords / intervals played (trained / tested) will start from a random position on the piano. A random example would be an A2 Major chord on one press and on the next press it would be a D#3 Major chord etc.

Model Classes:

Overview: The general patterns (scale / chord / interval) will be stored as a List of integers that corresponds with the intervals for the pattern in a concrete implementation of the class APatternDefinition. Example, Major scale -> [0, 2, 4, 5, 7, 9, 11, 12], Major chord -> [0, 4, 7], Major 3rd Interval -> [0, 4]. These definition instances will be created on application startup and will be stored into global Lists.

The specific patterns (A Major chord, C# minor scale, etc.) will be an instance of a concrete implementation of the abstract class APlayablePattern. They will contain a List of MusicNote objects and will have an implementation of the Play() method that will play the pattern. These instances will not be initially created or stored globally. Instead they will be created as they are needed by the application when the user selects to hear a scale / chord / interval.

* MusicNote – Holds a reference to the note name and associated sound file
  + System.Media.SoundPlayer
    - Audio file
    - Play() method already implemented
  + String note
  + Int midiNoteNumber
* Abstract APatternDefinition
  + String name – ex. Major chord
  + List<int> intervals
  + List<int> fingerPositions
  + int noteRange
* MusicScaleDefinition extends APatternDefinition
* MusicIntervalDefinition extends APatternDefinition
* MusicChordDefinition extends APatternDefinition
* Abstract APlayablePattern
  + String specificName – ex. A# minor chord
  + List<MusicNote> notes
  + List<int> fingerPositions
  + Abstract void Play()
* MusicScale extends APlayablePattern
* MusicInterval extends APlayablePattern
* MusicChord extends APlayablePattern

Utility Classes:

* NoteFinder
  + MusicNote FindNoteByName(string noteName)
  + Button FindButtonByNote(string noteName)
* PianoPlayer
  + Void PlayPianoNote(Button button) – Changes piano key color
  + Void PlayPianoNote(MusicNote note) – Changes piano key color
  + Void PlayPianoScale(MusicScale scale)
  + Void PlayPianoInterval(MusicInterval interval)
  + Void PlayPianoChord(MusicChord chord)
* RandomLoader
  + MusicScale LoadRandomScale()
  + MusicInterval LoadRandomInterval()
  + MusicChord LoadRandomChord()

Events:

* ClickPianoButton(Sender sender) – Plays a music note
* PressComputerKeyboardKey(Key key) – Plays a music note
* ChangeCheckboxHideKeyboard(Sender sender) – Hides the keyboard
* ChangeCheckboxHideKeyboardLabels(Sender sender) – Hides the note names on the keyboard
* ChangeCheckBoxRandomize(Sender sender) – Turns random mode on / off

UI Components:

* Piano – List of buttons wrapped in a panel
* HideKeyboardCheckbox
* HideKeyboardLabelsCheckBox
* RandomizeCheckBox
* TabPanel
  + List<Button> for each tab (all scales / chords / intervals)
  + MusicScalesTab
  + MusicIntervalsTab
  + MusicChordsTab

Global Lists:

* List<Button> pianoButtons
  + All the piano buttons on the UI screen from C2 – B4
* List<MusicNote> musicNotes
  + Contains all music notes from C2 – B4
* List<MusicScaleDefintion> musicScales
  + All the scales defined in our app
* List<MusicIntervalDefintion> musicIntervals
  + All the intervals defined in our app
* List<MusicChordDefintion> musicChords
  + All the chords defined in our app
* List<Button> musicScaleButtons
  + The list of musicScales rendered into buttons
* List<Button> musicIntervalButtons
  + The list of musicIntervals rendered into buttons
* List<Button> musicChordButtons
  + The list of musicChords rendered into buttons

Global Defines:

* Define all intervals so they can be used in place of int’s
  + public const int MINOR\_SECOND = 1;
  + public const int MAJOR\_SECOND = 2;
  + Etc.

Assets:

* Short audio clip for every note played on a piano from C2 – B4

References:

* Computer keyboard -> piano note layout



* Midi note numbers

