Auditory Ace



A Godot Cochlear Implant Practice Application

Original developers,

Darya Haines, Dustin Huynh, Jordan Nguyen, Layla Smith, Nihar Koppolu, Scott Thorkelson, Sienna Day

[Overview 5](#_gdq31s15p3hf)

[UI Concepts 5](#_y15zgtcbyigf)

[Color Choices 5](#_fx1w8ra4orvq)

[Fonts 6](#_8e9sjqan5fr0)

[Icons 6](#_99d0jf7lqh6i)

[Screen Examples 6](#_sdku937yqto7)

[Home Page 7](#_jkytxaq3xdu0)

[Main Menu Code Documentation 8](#_nhxxfs4bl2cz)

[1. Title and Logo 8](#_5y1fm9e4wwzq)

[2. Navigation Buttons: 8](#_pz2iw38rg1dx)

[3. Functionality Buttons: 8](#_v5hctlym6twm)

[4. Visual Elements: 9](#_f0rsjuecf3n6)

[5. Interactivity: 9](#_p45cyyffp00o)

[Intro Screen Code Documentation 9](#_skhcm65fur2m)

[1. Title Label (AppTitle): 9](#_915p87iqeuhi)

[2. Image (Img9060): 9](#_bn3ioz2tbqm)

[3. Background (ColorRect): 9](#_461a5wk4d6ln)

[4. Timer: 9](#_nnmzchekm6bu)

[5. Acknowledgment Label (Label): 9](#_ooung9kr4x8e)

[6. Script: 10](#_avq2pqhx5pnu)

[7. Signal Connection: 10](#_vnj4vzggp3s3)

[Exercises 10](#_iyujxyji98mc)

[Important Concepts: 10](#_h1j6sf14i9lk)

[Exercise One 11](#_yqi16b4kt8bv)

[In-app Description: 11](#_nw2jbdwm6n8)

[1. ExerciseOne.gd 11](#_886t1c38iqel)

[Overview: 11](#_yxi7k8xv11cb)

[Key Features: 12](#_pmith5ft25bt)

[Usage: 12](#_8a0d189bb77x)

[Functions: 12](#_rhtho0bnkb6d)

[2. ExerciseOne.tscn 13](#_82ff3pmu10n9)

[Overview: 13](#_xbi39hth4gmj)

[Components: 13](#_fvy7o8b5xnpp)

[Usage: 13](#_x2n0ea6qc4z6)

[Exercise Two 13](#_jtntco2lkde7)

[In-app Description: 14](#_a0f4hjqxgyw6)

[1. Exercise\_two.gd 14](#_601wjj7k3g0q)

[Description: 14](#_84wojcitxdzh)

[Variables: 14](#_z876b6cwghmu)

[Other Variables: 15](#_1aogl8ebvtgk)

[2. Exercise\_two.tscn 15](#_1zkyr8hpa2py)

[Description: 15](#_kzfe0m8dn0o5)

[Nodes: 15](#_mfazq6l65rdz)

[Pre-Exercise Screens 16](#_ko7e0dk64t9)

[Overview 16](#_d57iuy52rnyj)

[Node Structure: 16](#_uw55j57dtsig)

[External Resources: 16](#_t27fuu18z12i)

[Styles and Themes: 16](#_d43q9np563g8)

[Functionality: 16](#_sr7pyirtdvv8)

[UI Components: 16](#_vfmsq3w2mbca)

[Post-Exercise Screen 18](#_ctuxr43zvbpg)

[GDScene File: post\_exercise\_screen.gd 18](#_5eb624gsh3gq)

[Variables: 18](#_6j612z34q0v5)

[Methods: 18](#_l93kxwr1chns)

[GDScene File: post\_exercise\_screen.tscn 18](#_hwz8p1qlkamr)

[Nodes: 18](#_w24f581snvyp)

[Connections: 19](#_g8jrczd82x8q)

[UI Elements: 19](#_ytff1wulk0r9)

[Styling: 19](#_tl6pce2xknbu)

[Layout: 19](#_z2lmrcnp466q)

[How to Create an Exercise 20](#_kx2315nd19i2)

[Step 1: Plan the Exercise 20](#_ay7ocmnii4r3)

[Step 2: Set Up the Scene 20](#_is1o2r3yi3uz)

[Step 3: Implement Exercise Logic 21](#_dtr1mc75arwz)

[Step 4: Integrate Audio and Visual Assets 22](#_c2wcgbrq8hdu)

[Step 5: Test and Debug 22](#_i04pi5ujub7c)

[Step 6: Add in Pre- and Post- Exercise Screens 22](#_hifiun9dzjgj)

[Step 7: Review and Iterate 23](#_med0uovqudnm)

[Statistics 23](#_pwqoyqhhy1sh)

[Overview 23](#_cmuzqs83mxdp)

[Code Documentation 24](#_q0avws4m6mlz)

[Overview: 24](#_lpszxlp5esx0)

[Files: 24](#_5xgyfr6sjj67)

[StatisticsTesting.gd 24](#_4tfh85hlqh17)

[Purpose: 24](#_zf0o80vibo2r)

[Functionality: 24](#_c2mzofx1w38k)

[Structure: 25](#_pr4310g5lst9)

[StatisticsTestingScene.tscn 25](#_f2r97o1wyy8h)

[Purpose: 25](#_i4z49we3duc7)

[Components: 25](#_mttmaffxy3hd)

[● Nodes: 25](#_sciwbuxnt7ih)

[● Connections: 25](#_o4ju13khaw5x)

[Layout: 25](#_4ss7qpdnozq1)

[Functionality: 25](#_5xkou2rln5bi)

[1. Searching Entries: 25](#_w79iy2yu304y)

[2. Generating Random Entries: 26](#_89hx8cbdts9f)

[Scoring System 26](#_c9scozn91l19)

[Database 27](#_pdkoh0opf1zn)

[Usages: 28](#_voustasovt0x)

[Adding an entry to the database 28](#_tfc1uv3j1416)

[Search/retrieve entries 28](#_6w2g3879zf62)

[Loading a setting 29](#_eo2kuze7erq1)

[Updating a setting 29](#_eetnvtkzm7wt)

[CSV Files 29](#_frgd92f88ifx)

[Design 30](#_qoub7q5vhbkd)

[Word List CSV Example 30](#_h2v7dowgi8pt)

[Sentence List CSV Example 30](#_6wvt4p2cb0mo)

[Documentation for WordListManager.gd 30](#_bt17g1njbcwi)

[Overview: 30](#_cin2v2icjkrb)

[File Structure: 30](#_blkvt59x208o)

[Usage: 31](#_rl95gfi45yqs)

[Functions and Methods: 31](#_sgcjussln4x2)

[Variables: 31](#_5hwj6q656p3b)

[Dependencies: 32](#_2wpswjo2csfc)

[Error Handling: 32](#_vpc6e2ro78jk)

[Example Usage: 32](#_f3y7du38ifvj)

[How To Add A New Word List Type 33](#_dqrdra26h4hc)

[Notes: 34](#_je5oq6x69ecj)

[SentenceListManager.gd Documentation 34](#_ue1yrfs8s77t)

[Overview 34](#_tqd47n10wfzd)

[Class Structure 34](#_x3qrzmpxikz5)

[SentencePair 34](#_b87955pksp0x)

[WordListType (Enum) 34](#_ki8llkw117uc)

[Properties 35](#_x1e1lctq4auo)

[Methods 35](#_7oru9hax6ssm)

[Example GDScript Usage In An Exercise 35](#_1dpdyb3kbvvt)

[Usage 36](#_g98q5kpvc308)

[Important Notes 36](#_yyzlakq8j8eo)

[Sound 37](#_o424gghmhvhu)

[Volume Check 37](#_pxr4gdnz1pq9)

[Overview: 37](#_mrec1425jp1i)

[Scene Information: 37](#_tyzbb2zcwtp1)

[External Resources: 38](#_ffmi909cf2ip)

[StyleBoxFlat Subresources: 38](#_43x7rw9v7buz)

[Nodes and Components: 39](#_lxrooerlmcm1)

[Functionality: 40](#_d6xv4fcif0nz)

[Text-to-Speech 40](#_d2qtx9d2ghah)

[Overview 40](#_h1p6az8o7gjl)

[Usage 40](#_irpzawgq3c7d)

[Invoking TTS 40](#_rvhqleou0458)

[Changing TTS Voice 40](#_j7ncz8t890ga)

[Changing TTS Volume 40](#_mz107p479ipr)

[Sound Effects 41](#_gs52xypm475x)

[Overview 41](#_dpxhtqr578mt)

[Usage 41](#_40zo0arwdb6o)

[Playing Sound Effect 41](#_vnb6ersboo2j)

[Changing Sound Effect Volume 41](#_akbr0zg26940)

[Background Noise 41](#_8ue5upyivfpw)

[Overview 41](#_prtjemsfi91w)

[Usage 41](#_ghjdi7pc1zhm)

[Load Specified Background Noise 41](#_wya0czn7884j)

[Changing Background Noise Settings 42](#_vevsm8gl4e4d)

[Playing/Stopping Background Noise 42](#_vopxi7d1l4ra)

[Help Page 42](#_6qp20grzbnjp)

[Introduction 43](#_iyvlpiy79gwo)

[Links to important documentation 43](#_nuuwuz5vl9b2)

[List of links: 43](#_j4kb9i9gd32m)

[Email Address 43](#_42l8ta6m56cf)

# Overview

Auditory Ace is an app designed for people of all ages who use a Cochlear Implant and was originally designed by the Deviant Developer’s Dream Capstone Team at Portland State University during the Fall 2023 / Winter 2024 capstone. The app serves as a specialized platform, offering users a unique opportunity to hone their auditory skills by practicing sound recognition through their implants. With a user-centric approach, Auditory Ace provides diverse sound exercises, encompassing simple tones and everyday sounds. The app's inception within the academic environment showcases the team's commitment to leveraging technology for positive impact, as it empowers Cochlear Implant users to engage in targeted practice sessions, fostering a richer and more nuanced auditory experience.

# UI Concepts

Auditory Ace contains many different artistic choices that were made to better create an enjoyable experience for each user of the application. This documentation serves as a comprehensive guide, detailing the intricacies of the thoughtfully designed user interface (UI) and user experience (UX) elements that define the app's functionality. The team's commitment to creating a seamless and accessible interface is evident throughout the design, ensuring that users of all age groups can engage effortlessly in sound recognition practice. From the implementation of customizable settings to the strategic integration of gamification elements, this documentation provides an exploration of the design principles governing Auditory Ace's UI/UX, facilitating a thorough understanding for future development teams aiming to build upon and enhance the application.

## Color Choices

Auditory Ace used color-blind-friendly colors in its design so that more people would be comfortable using the app. Below are the RGB codes for each color option used in the application as well as their color names and where they were used in the design.

| RGB Code | Color Name | Usage |
| --- | --- | --- |
| 000000 | black | font and icons |
| a1d2f1 | baby blue color | background |
| daefda | white | containers on page |

## Fonts

Auditory Ace used the DM Sans 24pt Medium font choice in the screen designs. While exercises may use a different font depending on its theme, this font is the main font for the application.

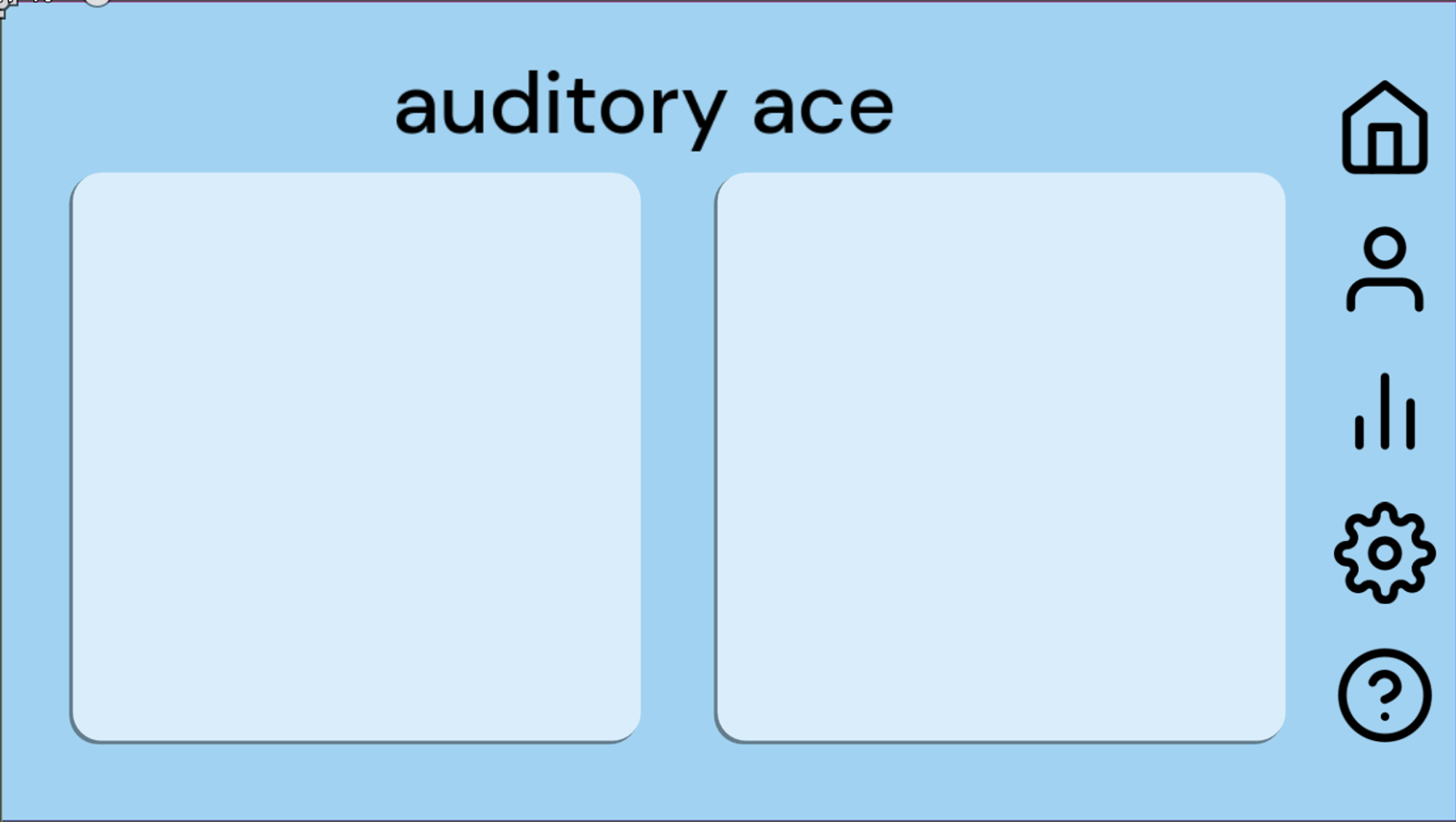
## Icons

Icons can be found in the icons.zip file in the code database

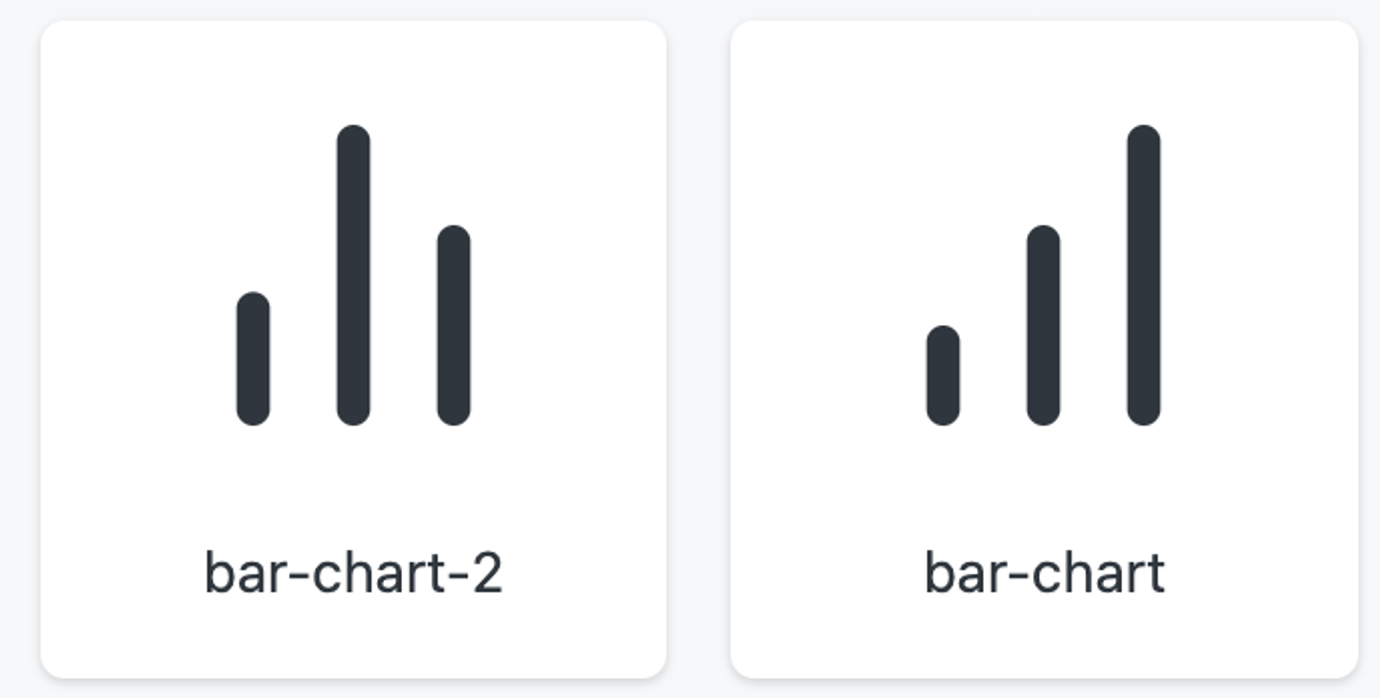
Flash card icons used with credited permission from [2,414 Free icons of flash cards](https://www.flaticon.com/free-icons/flash-cards) .

## Screen Examples

Here are a few examples of what the early screens looked like for Auditory Ace.



Home Screen



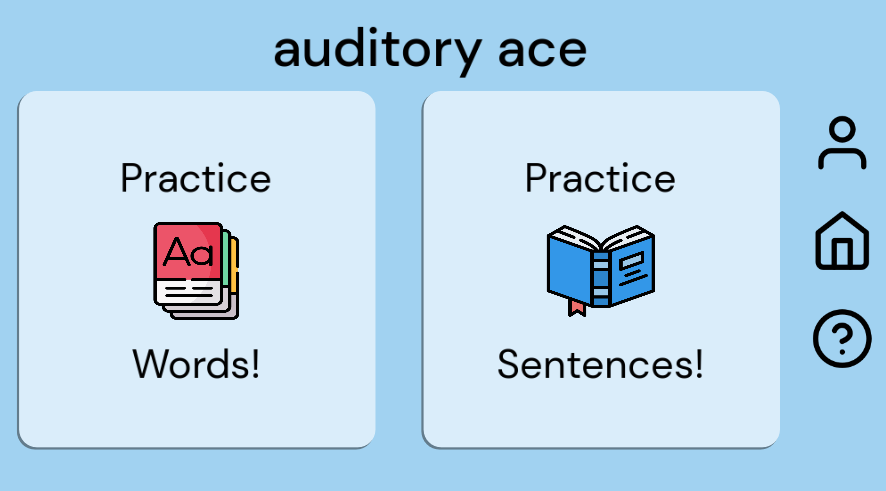
Statistics Charts

# Home Page

Auditory Ace has a unique homepage where you can access all exercises. All areas in the app have a sidebar where you can navigate between the homepage, your statistics, and settings.

The home page for Auditory Ace should be clean, with only two exercises in a row. When adding more exercises, add a scroll functionality to not clutter up the home screen. There is also a sidebar that allows the user to switch between screens and should always be the same.

For the exercise options, they should all have the same shape, color, and formatting. An image should also be added to make the screen look better from a user perspective using an icon from <https://www.flaticon.com/free-icons/flash-cards>.



Home Screen as of 3/3/23

## Main Menu Code Documentation

The main menu serves as the primary interface for users to navigate through the application or game. It provides access to essential features and functionalities. Here's an overview of the main menu:

### Title and Logo

The main menu typically displays the title of the application or game along with a logo or visual identifier. This helps users recognize the brand or purpose of the software.

### Navigation Buttons:

* The main menu contains various buttons that allow users to navigate to different sections or screens within the application. These buttons are often labeled with descriptive text or icons to indicate their function.

### Functionality Buttons:

* Profile: Allows users to access their profile information or settings.
* Home: Navigates users back to the main screen or starting point of the application.
* Help: Provides assistance or guidance to users, such as instructions or FAQs.
* Exercise One and Exercise Two: These buttons lead users to specific exercises or tasks within the application.

### Visual Elements:

* The main menu may include visual elements such as background colors, images, or animations to enhance its aesthetic appeal and reinforce the application's theme or branding.

### Interactivity:

* Buttons in the main menu are interactive, responding to user input such as mouse clicks or touch gestures. They trigger actions like scene transitions or opening specific functionalities within the application.

## Intro Screen Code Documentation

The intro screen, or introductory screen, is the initial interface that users encounter when launching the Auditory Ace application. Its primary purpose is to provide a brief introduction or welcome message to users. Here's an overview of the components within the intro screen:

### Title Label (AppTitle):

* Displays the application or game title "uditory ace".
* Positioned at the center of the screen to ensure visibility.
* Utilizes the DMSans font with a font size of 80 for clear readability.
* Aligned horizontally and vertically for aesthetic presentation.

### Image (Img9060):

* Represents an image relevant to the application or game.
* Positioned at a specific location for visual appeal.
* Scaled and rotated for proper fitting within the layout.

### Background (ColorRect):

* Provides a colored background to the intro screen.
* Utilizes a light blue color (RGB: 0.631373, 0.823529, 0.945098) for a soothing and welcoming appearance.
* Covers the entire screen to ensure uniformity.

### Timer:

* A timer set to trigger after 2 seconds (wait\_time = 2.0).
* Configured as a one-shot timer (one\_shot = true) to trigger only once.
* Automatically starts upon the intro screen's initialization (autostart = true).

### Acknowledgment Label (Label):

* Displays an acknowledgment message "Made by Devious Developers".
* Positioned below the title for credit attribution.
* Utilizes the JosefinSans-Regular font with a font size of 20 for subtlety.

### Script:

* Linked to an external script file (intro\_screen.gd) for handling logic and interactions.

### Signal Connection:

* Connects the timeout signal emitted by the Timer node to a method named \_on\_timer\_timeout.
* This signal connection triggers a transition to the main menu after the specified time interval.

The intro screen sets the tone for the user experience and creates a positive first impression, welcoming users to the application environment.



Intro Screen as of 3/10/24

# Exercises

Each exercise follows the same basic UI look as the main screen. Each exercise also uses the same correct/incorrect toolbar at the top of the screen to indicate where the user is in the exercise.

### Important Concepts:

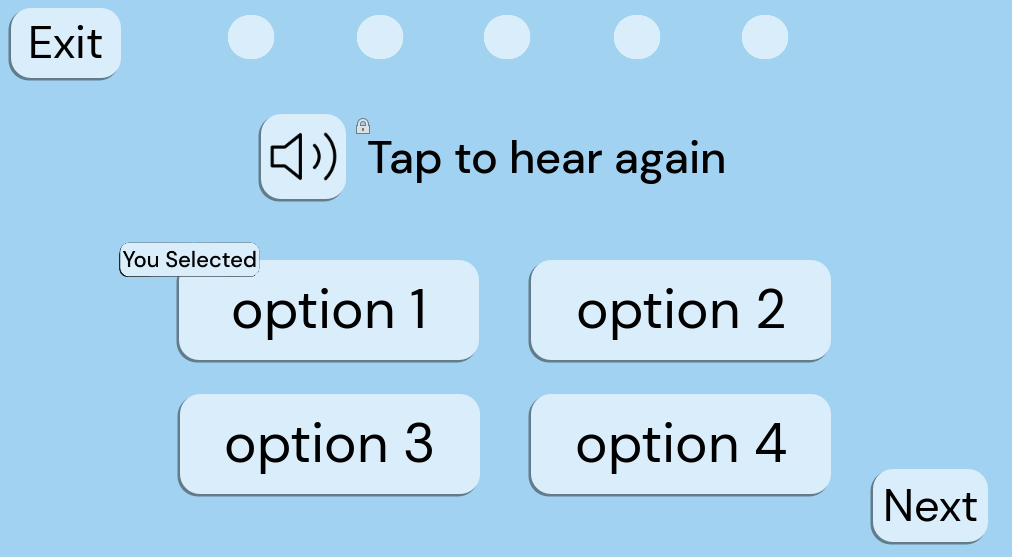
* The button labels must be easy to read and do not overlap with any other exercise elements.
* The exercise screens must not be too cluttered, as that hinders the learning experience for the users.

## Exercise One

This exercise is a four-word choice type of exercise. There are a predetermined number of questions in the exercise, and each question contains different words. The questions will play a sound (which can be replayed) and will give four options as to what the word could be. Then the user can select which word matches the sound. This allows the user to test their hearing for similar-sounding words.It is made up of two files, ExerciseOne.gd and ExerciseOne.tscn.

### In-app Description:

Let’s practice hearing and differentiating between similar-sounding words!



Example of Exercise 1

### 1. ExerciseOne.gd

#### Overview:

This script controls the behavior and logic of the first interactive exercise within the application. It manages user interactions with buttons, word generation, scoring, and progression through rounds.

#### Key Features:

* Word generation and display on buttons.
* User interaction handling for selecting words.
* Logic for determining correct answers.
* Handling game progression, including rounds and completion.

#### Usage:

1. Attach this script to a Node2D in your scene hierarchy.
2. Ensure that the necessary UI elements are properly linked in the scene.
3. Customize the script variables, such as the maximum number of rounds and the word lists.
4. Run the scene to allow users to interact with the exercise.

#### Functions:

* \_ready(): Initializes the exercise when the node enters the scene tree.
* onButton1Pressed(), onButton2Pressed(), onButton3Pressed(), onButton4Pressed(): Handlers for button presses.
* onSoundButtonPressed(): Handles sound button press to play correct word audio.
* onNextButtonPressed(): Handles the next button press to progress or end the game.
* onExitButtonPressed(): Handles exit button press to return to the previous screen.
* buttonLogic(buttonNum): Logic for processing button presses and determining correctness.
* generateWords(): Generates new word options for the buttons.
* checkCorrect(pressedWord, correctWord): Checks if the pressed word matches the correct word.
* changeNextStar(correctIncorrect, numRounds): Updates the visual indicator for correct and incorrect answers.
* buttonColorChange(colorBool): Changes button colors based on correctness and game state.
* gameDone(): Handles completion of the game and additional functionalities (not fully implemented).

### 2. ExerciseOne.tscn

#### Overview:

This scene file defines the layout and hierarchy of UI elements for Exercise One. It includes buttons, text elements, and visual indicators necessary for user interaction.

#### Components:

* Background: Node2D serving as the background for the exercise.
* ColorRect: ColorRect providing a colored background for text.
* TopText: Label displaying text instructions or feedback.
* Button: Node2D containing multiple Button nodes for user selection.
* SoundButton, ExitButton, NextButton: Buttons for specific actions within the exercise.
* Node2D: Container for visual elements like stars and indicators.
* Star1 to Star5: Visual indicators representing user performance.
* RoundTimer: Timer node for tracking time per round.
* SelectedIndicator: Visual indicator showing the selected word.

#### Usage:

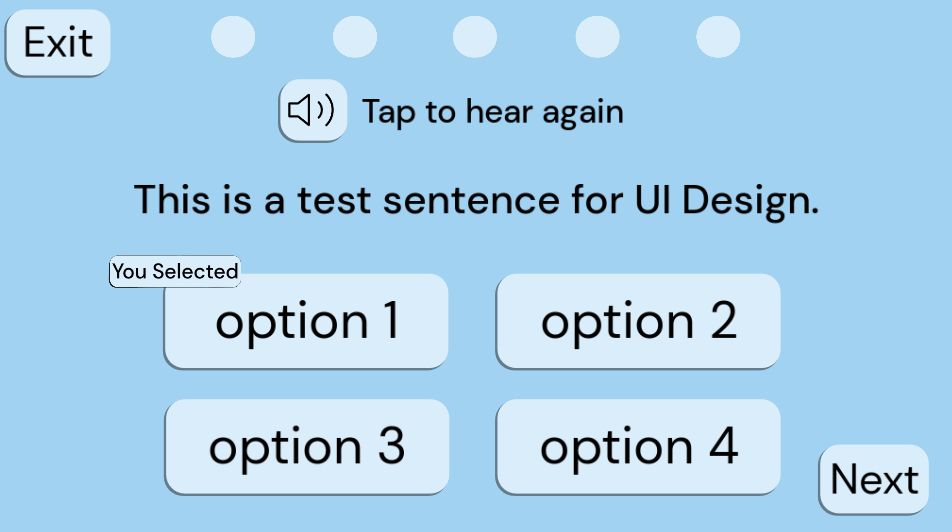
1. Open this scene file in the Godot editor.
2. Customize UI elements and their properties as needed.
3. Ensure that signals from buttons are connected to the appropriate methods in the script (ExerciseOne.gd).
4. Run the scene to allow users to interact with the exercise.
5. This documentation provides an overview of the provided files and their functionalities. Further customization and integration may be required based on specific project requirements.

## Exercise Two

This exercise is a sentence type of exercise. There are a predetermined number of questions in the exercise, and each question contains different words in sentences. The questions will play a sentence sound (which can be replayed) along with displaying the sentence minus a word and will give options as to what the missing word could be. Then the user can select which word matches the missing word’s sound in the sentence. This allows the user to test their hearing for similar-sounding words in sentence structures.

### In-app Description:

Let’s practice hearing and differentiating between similar-sounding words in sentences!



Example of Exercise 2

### Exercise\_two.gd

#### Description:

This GDScript file contains the logic for Exercise Two in the application. It manages the user interface elements, rounds, correct answers, and other game functionalities.

#### Variables:

* buttonOne: Reference to the first button node.
* buttonTwo: Reference to the second button node.
* buttonThree: Reference to the third button node.
* buttonFour: Reference to the fourth button node.
* nextButton: Reference to the next button node.
* starOne to starFive: References to star nodes representing the user's progress.
* roundTimer: Reference to the timer node.
* topText: Reference to the text node displaying instructions or feedback.
* selectedIndicator: Reference to the indicator node displaying the selected button.
* display\_sentence: Reference to the node displaying the current sentence to the user.

#### Other Variables:

* numRounds: Current number of rounds.
* maxNumRounds: Maximum number of rounds.
* correctWord: Current correct word.
* replayMode: Boolean indicating whether the game is in replay mode.
* soundIcon: Preloaded image for the sound icon.
* numCorrect: Number of correct answers.
* sentence: The current sentence being displayed to the user.

### Exercise\_two.tscn

#### Description:

This scene file defines the layout and structure of Exercise Two's user interface. It includes buttons, timer, feedback elements, and other visual components required for the game.

#### Nodes:

* Button: Container node for buttons.
  + Button1 to Button4: Individual buttons for user interaction.
  + NextButton: Button to proceed to the next round.
* Node2D: Container node for visual elements.
  + Star1 to Star5: Star nodes representing the user's progress.
  + Background: Container node for background elements.
    - hear\_again\_text: Node displaying instructions or feedback text.
    - display\_sentence: Node displaying the current sentence to the user.
  + SelectedIndicator: Node indicating the selected button.
* RoundTimer: Timer node for tracking round time.

This scene file serves as the visual representation of Exercise Two and is linked with the corresponding GDScript file (exercise\_two.gd) to provide functionality.

## Pre-Exercise Screens

### Overview

The pre-exercise screens for exercise one and exercise two allow the user to select the word list they want to practice, a text to speech voice, if they want background noise as well as its volume level.

### Node Structure:

Both screens contain nodes for UI elements like buttons, labels, and containers.

The first exercise screen has a more complex structure with option buttons and a VBoxContainer, while the second exercise screen is simpler, focusing on navigation.

### External Resources:

Both screens use external resources for fonts and icon textures.

### Styles and Themes:

The first exercise screen defines custom styles for buttons and backgrounds using StyleBoxFlat resources, while the second exercise screen does not.

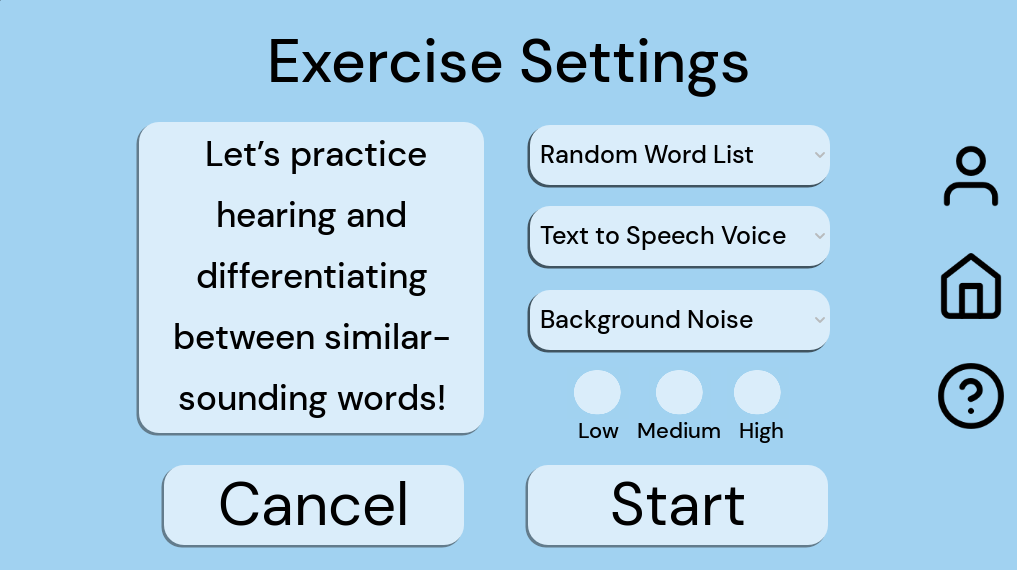
### Functionality:

Both screens provide basic functionality for navigation and button actions

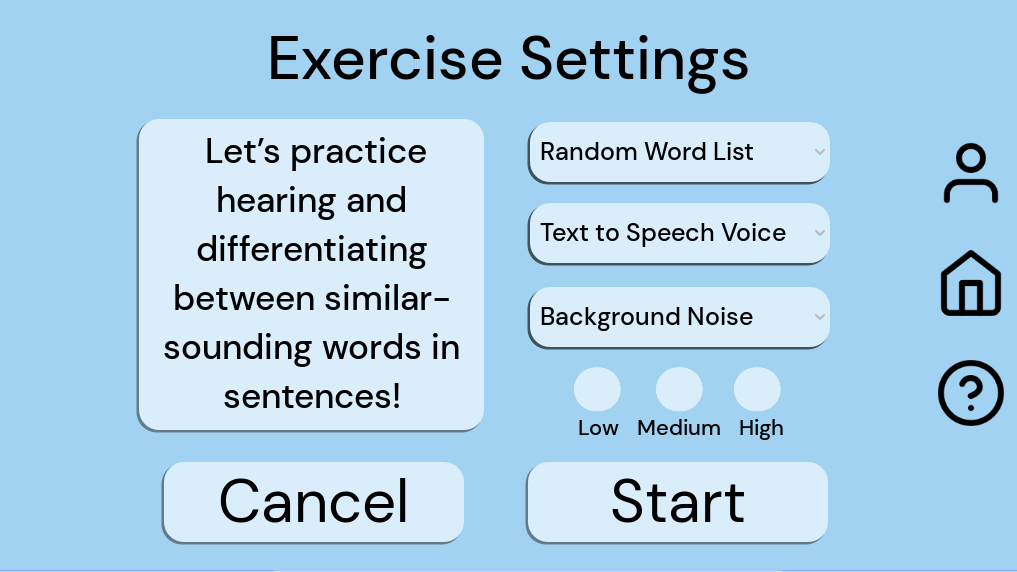
### UI Components:

Both screens utilize buttons, labels, and other basic UI elements.

The first exercise screen includes additional components like option buttons, making it more feature-rich compared to the second exercise screen.



Exercise One Pre-Exercise Screen



Exercise Two Pre-Exercise Screen

## Post-Exercise Screen

### GDScene File: post\_exercise\_screen.gd

This GDScript file defines the behavior and functionality of the post-exercise screen in a game or application. It extends the Node2D class and includes methods for handling button presses and initializing UI elements with data. Here's a breakdown of its components:

#### Variables:

* backscene: This variable stores a reference to the previous scene, likely for navigation purposes.

#### Methods:

* \_on\_cancel\_pressed(): Handles the press event of the "Cancel" button, changing the scene to pre\_exercise\_screen.tscn.
* \_on\_done\_pressed(): Manages the press event of the "Done" button, changing the scene to main\_menu.tscn.
* \_on\_profile\_pressed(): Deals with the press event of the "Profile" button, changing the scene to profile.tscn.
* \_on\_help\_pressed(): Manages the press event of the "Help" button, changing the scene to help.tscn.
* \_on\_home\_pressed(): Handles the press event of the "Home" button, changing the scene to main\_menu.tscn.
* \_ready(): Initializes UI elements with data such as score, word list text, background volume level, and accuracy.

### GDScene File: post\_exercise\_screen.tscn

This Godot scene file defines the layout and appearance of the post-exercise screen. It includes various UI elements such as buttons, labels, and containers. Here's an overview of its structure:

#### Nodes:

* Exercise 1 Settings: A node possibly holding settings related to Exercise 1.
* Background: Node containing visual elements like color rectangles.
* UI: Node containing UI elements like labels and buttons.
* Buttons: Node containing button elements.
* Globals: Node possibly storing global variables or references.

#### Connections:

* Establishes signal connections between UI elements and methods defined in the associated script file.

#### UI Elements:

* Labels: Display information such as game title, score, accuracy, word list practiced, and background sound level.
* Buttons: Perform actions like canceling, finishing, accessing profiles, getting help, and returning to the main menu.

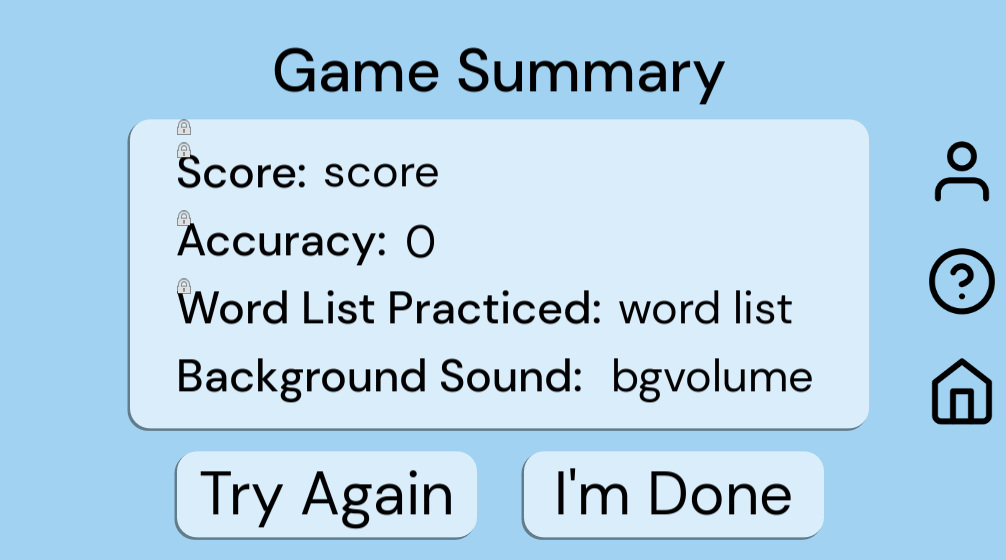
#### Styling:

* Style boxes are defined to customize the appearance of UI elements, including colors, corner radius, and shadows.

#### Layout:

* UI elements are organized within containers such as VBoxContainer for proper alignment and spacing.

This combined documentation provides insight into both the functionality and visual layout of the post-exercise screen in the game or application. It describes how the script interacts with UI elements to manage user interactions and scene transitions.



Example of post exercise screen

## How to Create an Exercise

Here is a step-by-step walkthrough on how to create a simple exercise for Auditory Ace using GDscript in Godot.

### Step 1: Plan the Exercise

1. **Define Exercise Objective**: Clearly define the purpose and objective of the exercise. Understand what skills or knowledge you want the user to acquire or practice.

2. **Design User Interface**: Sketch or design the user interface layout for the exercise. Determine the elements, such as buttons, labels, and graphics, needed to create the exercise interface.

### Step 2: Set Up the Scene

1. **Create a New Scene**: In Godot, create a new scene.tscn for the exercise. This scene will contain all the elements required for the exercise.

2. **Add Nodes**: Add necessary nodes to the scene, such as buttons, labels, color rectangles, and any other UI elements needed for the exercise.

Example button code:

**[node name="Button1" type="Button" parent="Button"]**

**offset\_left = 173.0**

**offset\_top = 268.0**

**offset\_right = 473.0**

**offset\_bottom = 370.0**

**theme\_override\_colors/font\_color = Color(0, 0, 0, 1)**

**theme\_override\_fonts/font = ExtResource("1\_1671l")**

**theme\_override\_font\_sizes/font\_size = 70**

**theme\_override\_styles/normal = SubResource("StyleBoxFlat\_8dnn1")**

**theme\_override\_styles/hover = SubResource("StyleBoxFlat\_yyuig")**

**text = "Cat"**

Example label code:

**[node name="AppTitle" type="Label" parent="Background/ColorRect"]**

**layout\_mode = 0**

**offset\_left = 163.0**

**offset\_top = 126.0**

**offset\_right = 943.0**

**offset\_bottom = 206.0**

**theme\_override\_colors/font\_color = Color(0, 0, 0, 1)**

**theme\_override\_fonts/font = ExtResource("13\_s6hby")**

**theme\_override\_font\_sizes/font\_size = 60**

**text = "Tap to hear again"**

**horizontal\_alignment = 1**

**vertical\_alignment = 1**

Example Color Code:

**[node name="ColorRect" type="ColorRect" parent="Background"]**

**offset\_right = 1024.0**

**offset\_bottom = 576.0**

**color = Color(0.631373, 0.823529, 0.945098, 1)**

3. **Organize Nodes**: Organize the nodes within the scene hierarchy according to their relationships and functionality.

### Step 3: Implement Exercise Logic

1. **Extend Node2D Class**: Create a new script.gd file that extends the Node2D class. This script will contain the logic for the exercise.

2. **Define Variables**: Declare variables for nodes and other necessary data structures. Use @onready to initialize node references.

3. **Implement Callback Functions**: Implement callback functions for user interactions, such as button presses. These functions will handle user input and trigger appropriate actions.

4. **Define Game Logic**: Write functions to generate words, check answers, update UI elements, and manage game flow based on user input.

### Step 4: Integrate Audio and Visual Assets

1. **Prepare Audio Assets**: If the exercise involves audio, prepare audio files for words or feedback sounds. Load and play these audio files as needed in the exercise logic.

2. **Prepare Visual Assets**: Prepare any visual assets required for the exercise, such as icons, buttons, background images, and UI elements. Ensure they are appropriately sized and formatted.

### Step 5: Test and Debug

1. **Test Exercise Flow**: Playtest the exercise to ensure it functions as intended. Verify that user interactions trigger correct responses and UI updates.

2. **Debug Logic**: Debug any issues or errors encountered during testing. Use debugging tools provided by the game development environment to identify and fix problems.

### Step 6: Add in Pre- and Post- Exercise Screens

1. **Add in Pre-Exercise Screen:** Create a modified version of the existing pre-exercise screens that will fit in with the current exercise.

2. **Add in Post-Exercise Screen:** Add in the existing post-exercise screen to appear after the exercise is completed.

Post-Exercise Code:

**extends Node2D**

**func \_on\_cancel\_pressed():**

**get\_tree().change\_scene\_to\_file("res://Scenes/pre\_exercise\_one\_screen.tscn")**

**func \_on\_done\_pressed():**

**get\_tree().change\_scene\_to\_file("res://Scenes/main\_menu.tscn")**

**func \_on\_profile\_pressed():**

**get\_tree().change\_scene\_to\_file("res://Scenes/profile.tscn")**

**func \_on\_help\_pressed():**

**get\_tree().change\_scene\_to\_file("res://Scenes/help.tscn")**

**func \_on\_home\_pressed():**

**get\_tree().change\_scene\_to\_file("res://Scenes/home.tscn")**

### Step 7: Review and Iterate

1. **Review Feedback**: Gather feedback from users or testers who interact with the exercise. Use this feedback to identify areas for improvement or refinement.

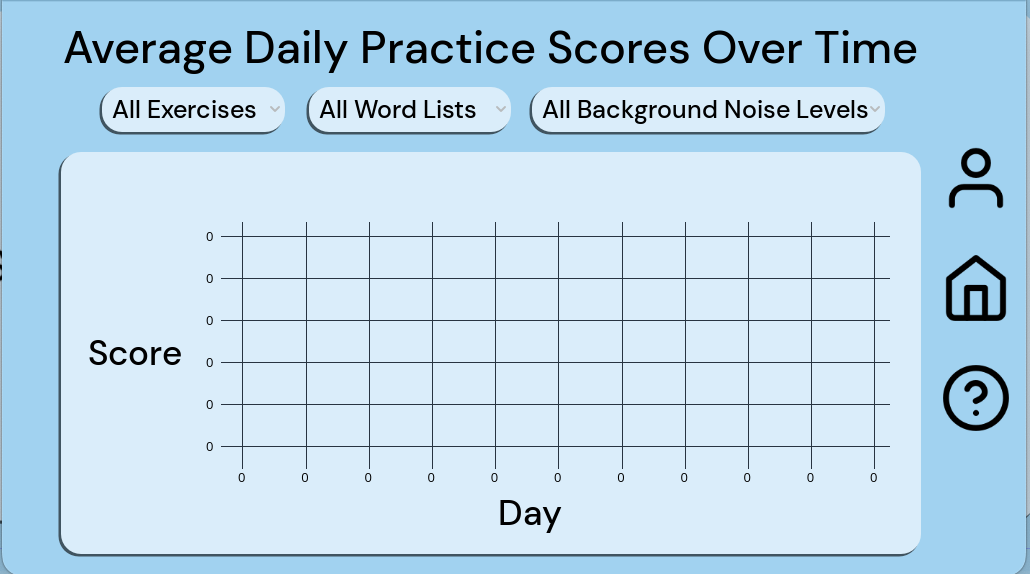
2. **Iterate on Design**: Iterate on the exercise design and implementation based on user feedback and testing results. Make necessary adjustments to improve the exercise experience.

By following these steps, you can create a well-designed and functional exercise within your game or application, providing users with an engaging and effective learning experience.

# Statistics

## Overview

The statistics appear on two types of screens: the screen at the end of an exercise and the statistics page. Everything for the exercise post-screen is in the exercise documentation. This section will detail the data used in the statistics as well as the code used in the statistics screen.



Statistics Screen as of 3/10/24

## Code Documentation

## Overview:

The Statistics Testing Module is a component of a larger application developed in the Godot Engine. It is designed to manage statistical data entries, allowing users to search through existing entries and generate random data for testing purposes.

## Files:

### StatisticsTesting.gd

#### Purpose:

* This GDScript file defines the functionality for managing statistical data entries.

#### Functionality:

* It contains methods for searching through existing entries and generating random entries.
* It establishes connections between UI elements and methods to trigger actions based on user interaction.

#### Structure:

* The script extends the Node2D class.
* It includes methods such as searchEntries() and generateRandomEntries() for performing specific actions.
* Connections between button signals and methods are established within this file.

### StatisticsTestingScene.tscn

#### Purpose:

* This scene file defines the structure and layout of the user interface (UI) for the Statistics Testing Module.

#### Components:

##### Nodes:

* + It contains various nodes representing different UI elements such as buttons, labels, and input fields.

##### Connections:

* + Signal connections between UI elements and methods defined in the StatisticsTesting.gd script are set up within this file.

#### Layout:

* The scene is organized hierarchically with nodes nested within each other to create the desired layout for the user interface.
* UI elements are positioned and styled according to the requirements of the module.

## Functionality:

### Searching Entries:

* Users can input search parameters such as date, background noise, sound, and exercise.
* Upon triggering the search action, the module retrieves matching entries from the database and displays them in the UI.
* Entries are displayed in a tabular format, showing date and corresponding scores.

### Generating Random Entries:

* Users can trigger the generation of random entries by clicking on a designated button.
* The module adds a specified number of random entries to the database, which can be used for testing purposes.

Note: Some functionality related to additional entry fields such as time, background noise, sound, and exercise is currently commented out, suggesting that it may be a work in progress or placeholder code for future implementation.

## Scoring System

The scoring system of Auditory Ace uses two metrics when evaluating the user’s score; accuracy and time. A score will be given for each individual word or phrase answered by the user which ranges between 0-1000. Incorrect guesses will result in zero points given. Applying background noise rewards the user with a score multiplier:

| **Background Noise Level** | **Score Multiplier** | **Max Score (For each entry)** |
| --- | --- | --- |
| None | 1.0x | 5000 |
| Low | 1.2x | 6000 |
| Medium | 1.4x | 7000 |
| High | 1.6x | 8000 |

Each exercise has a pre-determined grace period that allows the user to get full credit if they answer within a specific time frame. The scoring algorithm uses the following expression:

**# Subtract from 1000 the minimum between the following expressions:**

**# 5 \* sqrt(time after grace period)**

**# 150 \* log10(time after grace period)**

**score = 1000 - min((5 \* sqrt(timeTaken - gracePeriod)), (150 \* (log(timeTaken - gracePeriod) / log(10))))**

The following table provides sample scores based on time:

| Time (after grace period) | Score (no multipliers) |
| --- | --- |
| 0s | 1000 |
| 1s | 842 |
| 5s | 646 |
| 10s | 500 |
| 16s | 368 |
| 25s | 340 |
| 45s | 302 |
| 90s | 257 |
| 5 minutes | 178 |

# Database

Auditory Ace’s database manages the local database of the app using SQLite.

The database consists of two tables:

* Settings - Contains values for settings
* Name (CHAR) - The name of the setting referenced
* Sound (CHAR) - The sound that is outputted by the setting
* Volume (INT) - The volume at which the sound is played at
* Entries - Contains user statistics for each sound, each entry records information from exercises
* Date (TIME) - The date on which the entry was completed at
* Score (INT) - The user’s performance regarding the specific background noise level, word sound, exercise
* Time (FLOAT) - TBD (May be removed)
* BackgroundNoise (CHAR) - The volume level of the background noise
* Sound (CHAR) - The word sound that was practiced on
* Exercise (CHAR) - The exercise that the entry was recorded on

## Usages:

### Adding an entry to the database

**# score: int (Example: 0, 124, 2831, 5000)**

**# time (in ms): float (Example: 1041, 4137, 12042)**

**# bgNoise: string (Example: None, Low, Medium, High)**

**# sound: string (Example: MvN, TvP, SvF)**

**# exercise: string (Example: Exercise 1, Exercise 2)**

**Database.addEntry(score, time, bgNoise, sound, exercise)**

### Search/retrieve entries

**# searchEntries(): returns a db query result (an array of dictionaries) which contains all entries that match search result**

**# Days: string (Example: 1, 7, 30, 365, All)**

**# BGNoise: string (Example: None, Low, Medium, High, All)**

**# Sound: string (Example: owe, ear, e, All)**

**# Exercise: string (Example: Exercise 1, Exercise 2, All)**

**var Entries = Database.searchEntries(Days, BGNoise, Sound, Exercise)**

**# Processing entries**

**# Entries[x][y]**

**# x: array index of entries returned**

**# y: Dictionary for database (Date, Score, Sound, etc)**

**for n in range (0, entries.size(), 1): # Iterate through entry array**

**$TextElements/EntryTable/Date.text += Entries[n]["Date"]**

**$TextElements/EntryTable/Score.text += str(Entries[n]["Score"])**

**$TextElements/EntryTable/Time.text += str(Entries[n]["Time"])**

**$TextElements/EntryTable/BGNoise.text += str(Entries[n]["BackgroundNoise"])**

**$TextElements/EntryTable/Sound.text += Entries[n]["Sound"]**

**$TextElements/EntryTable/Exercise.text += Entries[n]["Exercise"]**

### Loading a setting

**# retrieveSetting(): returns an array which contains the setting specifications**

**# setting: string (Example: Default, Exercise 1 TTS, Exercise 2 BGNoise)**

**var Setting = Database.retrieveSetting(setting)**

**# Setting[0] contains the Sound**

**# Setting[1] contains the Volume**

**# Setting TTS settings after retrieving settings**

**TextToSpeech.Voice = Setting[0]**

**TextToSpeech.Volume = Setting[1]**

### Updating a setting

**# category: string (Example: Default, Exercise 1 TTS, Exercise 2 BGNoise)**

**# setting: string (Example: Sound, Volume)**

**# value: int, float (Example: 0, 1, 2.5, 12, 50)**

**Database.updateSetting(category, setting, value)**

# CSV Files

The word lists and sentence lists used by the exercises in this application originate from CSV files. These files contain the words and sounds in the word CSV as well as the addition of the sentences in the sentence CSV file.

## Design

The design of these CSV files depended wholly on what they would contain. The word list CSV contains a word and a sound on each line while the sentence list CSV contains a word, a sentence, and a sound on each line.

### Word List CSV Example

**Word, Word, Word, Word, Word Type, Sound**

**Loam, Loan, Loon, Loom, Noun, /m/**

**Mode, Node, Mood, Nude, Noun, /m/**

**Boom, Boon, Dune, Doom, Noun, /m/**

**Mice, Nice, Neice, Mince, Noun, /m/**

**Mull, Null, Mill, Nil, Noun, /m/**

**Tin, Tent, Team, Tom, Noun, /n/**

**Run, Rum, Pun, Bum, Noun, /n/**

**Fans, Dams, Yams, Bans, Noun, /n/**

To add to this file, simply append at the end of the CSV file following the above framework.

### Sentence List CSV Example

**Sentence, Word Type**

**Do you have a []?, Noun**

**Can you pass the []?, Noun**

**The [] is on the table., Noun**

To add to this file, simply append at the end of the CSV file following the above framework.

## Documentation for WordListManager.gd

### Overview:

This script is designed to be used within the Godot game engine environment. It serves the purpose of managing word lists by reading from CSV files and providing functionality to select random word sets based on specific word-sound pairs. The script prevents duplicate word selections during exercises.

### File Structure:

- File Name: WordListManager.gd

- Extension: GDScript (.gd)

### Usage:

1. Extend Node:

This script extends the Node class in Godot, allowing it to be added to the scene tree.

2. Enumerated Types:

- WordListType: Enumerates different word list types, including M vs N, S vs F, T vs P, Food, and Place word types.

3. Classes:

- WordSet: Represents a set of words including the correct word, similar words, word type, and associated sound.

### Functions and Methods:

1. \_ready():

- Initializes the script when the Node enters the scene tree. Loads word sets from CSV files.

2. loadWordSets(filePath: String, type: WordListType):

- Loads word sets from a CSV file and categorizes them based on the specified word list type.

3. getRandomWordSet(type: WordListType) -> WordSet:

- Retrieves a random word set from the specified word list type while ensuring no duplicates.

4. setWordListVar(chosen: int):

- Sets the chosen word list type based on an integer input.

5. calculateTimeScore(correct: bool):

- Calculates the score based on the time taken to answer the exercise correctly.

### Variables:

- mVnWordSets, sVfWordSets, tVpWordSets, foodWordSets, placeWordSets:

Lists storing word sets for each word list type.

- usedmVnWords, usedsVfWords, usedtVpWords, usedFoodWords, usedPlaceWords:

Lists storing used words for each word list type.

- chosenWordList:

Represents the selected word list type.

- score:

Current score of the exercise.

- bgLevel:

Background level variable.

- initialTime, finalTime:

Variables for calculating time taken for exercises.

### Dependencies:

- Relies on Time class provided by the Godot engine.

- Requires properly formatted CSV files containing word sets.

### Error Handling:

- Provides error handling for file opening failures, unknown word list types, empty word lists, and incorrect CSV formatting.

### Example Usage:

**func generateWords():**

**# Using WordListManager.gd**

**var wordSet = WordListManager.getRandomWordSet(WordListManager.chosenWordList)**

**correctWord = wordSet.correctWord**

**# Playing text for user**

**TextToSpeech.playText(correctWord)**

**# Changing text on buttons randomly**

**var randomIndex = (randi() % 4) + 1**

**var buttons = [buttonOne, buttonTwo, buttonThree, buttonFour]**

**var j = 0**

**for i in range(1,5):**

**if(randomIndex == i):**

**buttons[i-1].text = wordSet.correctWord**

**else:**

**buttons[i-1].text = wordSet.similarWords[j]**

**j += 1**

### How To Add A New Word List Type

It is very simple to add in a new word list type. There are four total sections you will need to add your new word list to. First, create a CSV file following the CSV documentation. Then add your word list type to the enum WordListType.

After that, add your file call in the \_ready() function following the same style as the others in that function. Example file call:

**loadWordSets("res://Word-Lists/m vs n.csv", WordListType.MVN) # M vs N word types**

After that, it is time to add your append call in the load function section shown below.

**match type:**

**WordListType.MVN:**

**mVnWordSets.append(wordSet)**

**WordListType.SVF:**

**sVfWordSets.append(wordSet)**

**WordListType.TVP:**

**tVpWordSets.append(wordSet)**

Now add your new word type to the randomizer function. As shown below, this is how you can match the word list type in the exercise call.

**match type:**

**WordListType.MVN:**

**wordSets = mVnWordSets**

**usedWords = usedmVnWords**

**WordListType.SVF:**

**wordSets = sVfWordSets**

**usedWords = usedsVfWords**

**WordListType.TVP:**

**wordSets = tVpWordSets**

**usedWords = usedtVpWords**

Finally, add your word type to the setWordListVar function section shown below:

**match chosen:**

**1:**

**chosenWordList = WordListType.MVN**

**2:**

**chosenWordList = WordListType.SVF**

**3:**

**chosenWordList = WordListType.TVP**

It is very important to follow the same style as the code already has so that your word list type can be used easily.

### Notes:

- Ensure CSV files are properly formatted to avoid parsing errors.

- Use appropriate integer values to select word list types when calling setWordListVar().

- The script assumes that the CSV file contains one successful word, three incorrect words, the word type, and the word sound on each line separated by a comma.

- The script provides a foundation for word list management but can be extended based on specific project requirements.

## SentenceListManager.gd Documentation

### Overview

SentenceListManager is a script designed to facilitate the management and utilization of sentence lists stored in CSV (Comma-Separated Values) format. It provides functionalities to load sentence-word type pairs from a CSV file, retrieve random sentence pairs based on specified word types, and ensures that duplicates are avoided during random selection.

### Class Structure

#### SentencePair

* A class representing a pair of a sentence and its associated word type.
* Properties:
  + sentence: String - The sentence string.
  + wordType: String - The type of word associated with the sentence.
* Constructor:
  + func \_init(sentence\_arg: String, wordType\_arg: String): Initializes a SentencePair instance with provided sentence and word type.

#### WordListType (Enum)

* An enumeration representing different word types.
* Enum Values:
  + NOUN
  + ADJ
  + PLACE
  + FOOD
  + VERB

### Properties

* nounSentencePairs: Array - Stores sentence pairs associated with nouns.
* adjSentencePairs: Array - Stores sentence pairs associated with adjectives.
* placeSentencePairs: Array - Stores sentence pairs associated with places.
* foodSentencePairs: Array - Stores sentence pairs associated with food.
* verbSentencePairs: Array - Stores sentence pairs associated with verbs.
* usedNounSentences: Array - Tracks used noun sentences to prevent duplicates.
* usedAdjSentences: Array - Tracks used adjective sentences to prevent duplicates.
* usedPlaceSentences: Array - Tracks used place sentences to prevent duplicates.
* usedFoodSentences: Array - Tracks used food sentences to prevent duplicates.
* usedVerbSentences: Array - Tracks used verb sentences to prevent duplicates.

### Methods

* func \_ready(): Called when the node is ready. Loads sentence pairs from a CSV file.
* func loadSentencePairs(filePath: String) -> void: Loads sentence pairs from the specified CSV file path.
* func getRandomSentencePair(wordType: String) -> SentencePair: Retrieves a random sentence pair based on the specified word type.

### Example GDScript Usage In An Exercise

**# Using WordListManager.gd to get a random word set**

**var wordSet = WordListManager.getRandomWordSet(WordListManager.chosenWordList)**

**var wordType = wordSet.wordType**

**# Using SentenceListManager.gd to get a random sentence pair with matching word type**

**var sentencePair = SentenceListManager.getRandomSentencePair(wordType)**

**var sentence = sentencePair.sentence**

**# Find placeholders in the sentence (assumed to be indicated by [])**

**var placeholders = sentence.find\_all("[")**

**var words = []**

**# Replace placeholders with generated words**

**for \_ in range(placeholders.size()):**

**words.append(wordSet.correctWord)**

**for i in range(placeholders.size()):**

**var placeholderIndex = placeholders[i]**

**sentence = sentence.insert(placeholderIndex + i \* 2, words[i])**

**# Playing text for user**

**TextToSpeech.playText(sentence)**

**# Shuffle words for buttons**

**words.shuffle()**

**# Changing text on buttons**

**for i in range(4):**

**buttons[i].text = words[i]**

### Usage

1. **Initialization**: Ensure that the SentenceListManager node is added to your scene.
2. **Loading Sentence Pairs**: Call loadSentencePairs(filePath) method to load sentence pairs from a CSV file. Provide the file path as an argument.
3. **Retrieving Random Sentence Pairs**: Call getRandomSentencePair(wordType) method to retrieve a random sentence pair based on the specified word type. Provide the word type as an argument (e.g., "Noun", "Adjective", etc.).

### Important Notes

* The script assumes that the CSV file contains one word per line along with its sound pairing and a sentence separated by a comma.
* The script provides a foundation for sentence list management but can be extended based on specific project requirements.
* Ensure that the CSV file follows the format where each line contains a sentence followed by a comma and its associated word type.
* Avoid modifying the script unless necessary, especially the structure and logic of the methods, to ensure proper functionality

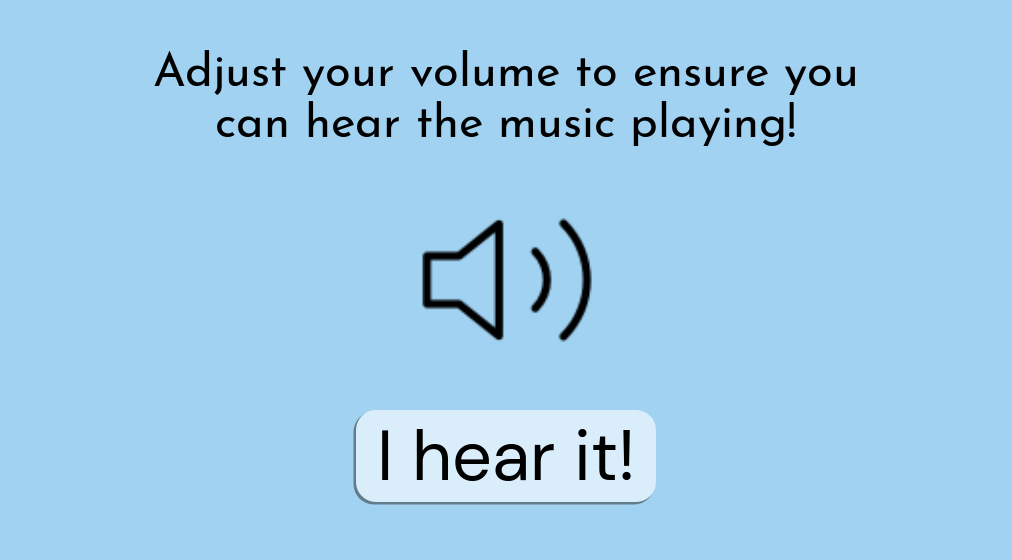
# Sound

This is the documentation covering everything sound related such as Text-to-Speech, Sound Effects, and Background Noise for the Auditory Ace application.

## Volume Check

### Overview:

This documentation outlines the structure and functionality of the Volume Check screen in a Godot project.The volume check sound can be found at [E-Lot - An ending theme. - Free Music Archive](https://freemusicarchive.org/music/e-lot/single/an-ending-theme/).



Volume Check as of 3/10/24

### Scene Information:

* Scene File: VolumeCheckScene.tscn
* Load Steps: 9
* Format Version: 3
* Unique Identifier: uid://ckjmnkj2oto1g

### External Resources:

1. Script Resource:

* Path: res://Scripts/volume\_check.gd
* ID: 1\_2vm7i

1. Font Files:

* DMSans Font:
  + Path: res://Fonts/DMSans\_24pt-Regular.ttf
  + ID: uid://bslnk4jwc4au1
* JosefinSans Font:
  + Path: res://Fonts/JosefinSans-Regular.ttf
  + ID: uid://dayil4h078ps1

1. Audio Stream:

* Path: res://Audio/E-Lot - An ending theme..mp3
* ID: uid://dpoy1i487e2j4

1. Texture 2D:

* Path: res://Icons/volume-2.svg
* ID: uid://cfisr5yjh2ob1

### StyleBoxFlat Subresources:

1. StyleBoxFlat\_dvl4f:

* Background Color: 0.854902, 0.929412, 0.980392, 1
* Corner Radius (All): 20
* Shadow Color: 0.129412, 0.129412, 0.129412, 0.486275
* Shadow Size: 1
* Shadow Offset: (-2, 2)

1. StyleBoxFlat\_2desg:

* Background Color: 0.854902, 0.929412, 0.980392, 1
* Corner Radius (All): 20
* Shadow Color: 0.129412, 0.129412, 0.129412, 0.486275
* Shadow Size: 1
* Shadow Offset: (-2, 2)

1. StyleBoxFlat\_w7w1w:

* Background Color: 0.854902, 0.929412, 0.980392, 1
* Corner Radius (All): 20
* Shadow Color: 0.129412, 0.129412, 0.129412, 0.486275
* Shadow Size: 1
* Shadow Offset: (-2, 2)

### Nodes and Components:

1. VolumeCheck (Node2D):

* Script: volume\_check.gd (ID: 1\_2vm7i)

1. ColorRect:

* Type: ColorRect
* Properties:
  + Offset Right: 1024.0
  + Offset Bottom: 576.0
  + Color: 0.631373, 0.823529, 0.945098, 1

1. Start Button:

* Type: Button
* Properties:
  + Layout Mode: 0
  + Offsets (Left, Top, Right, Bottom): 362.0, 420.0, 662.0, 512.0
  + Text: "I hear it!"
  + Font: DMSans\_24pt-Regular.ttf
  + Font Size: 70
  + Theme Overrides:
    - Normal Style: StyleBoxFlat\_dvl4f
    - Hover Style: StyleBoxFlat\_2desg
    - Pressed Style: StyleBoxFlat\_w7w1w

1. Subtext Label:

* Type: Label
* Properties:
  + Offsets (Left, Top, Right, Bottom): 112.0, 60.0, 912.0, 159.0
  + Text: "Adjust your volume to ensure you can hear the music playing!"
  + Font: JosefinSans-Regular.ttf
  + Font Size: 47
  + Horizontal Alignment: Center
  + Autowrap Mode: Word

1. Volume Icon Button:

* Type: Button
* Properties:
  + Offsets (Left, Top, Right, Bottom): 413.0, 190.0, 613.0, 390.0
  + Icon: volume-2.svg
  + Flat: true
  + Icon Alignment: Center
  + Expand Icon: true

1. Audio Player:

* Type: AudioStreamPlayer2D
* Properties:
  + Stream: "E-Lot - An ending theme..mp3"
  + Autoplay: true

### Functionality:

When the "Start" button is pressed, it triggers the \_on\_start\_pressed() function in the script volume\_check.gd, which changes the scene to the main menu (res://Scenes/main\_menu.tscn).

## Text-to-Speech

### Overview

TextToSpeech.gd manages all functions related to Text-to-Speech and is stored in the Database under Settings as ‘Default’.

### Usage

#### Invoking TTS

**# text: String (Example: Any text to be played via TTS)**

**TextToSpeech.playText(text)**

#### Changing TTS Voice

**# Retrieving list of voices**

**var Voices: Array[String] = TextToSpeech.getVoices()**

**# value: int (Index of voices in Voices array)**

**TextToSpeech.Volume = value**

**Database.updateSetting("Default", "Sound", value)**

#### Changing TTS Volume

**# value: int (Volume level)**

**TextToSpeech.Volume = value**

**Database.updateSetting("TTS", "Volume", value)**

## Sound Effects

### Overview

Audio.gd manages all functions related to Sound Effects

### Usage

#### Playing Sound Effect

**# text: String (Example: name of sound effect file excluding .mp3)**

**Audio.playFX(text)**

#### Changing Sound Effect Volume

**# value : int (Volume level of Sound Effect, ranges from -15 to 10)**

**Audio.changeFXVolume(value)**

## Background Noise

### Overview

Audio.gd manages all functions related to Background Noise. Four levels of background noise are used in the exercises: None, Low, Medium, High. The following table details the level (in decibels) each level of background plays at relative to TTS volume:

| Background Noise Level | Volume (relative to TTS volume) |
| --- | --- |
| Low | 20 db lower |
| Medium | 10 db lower |
| High | Same volume |

### Usage

#### Load Specified Background Noise

**# Intended to be called after entering exercise scene**

**# Loads specific noise file and volume into object**

**# noise: String (Example: name of the background noise file excluding .mp3)**

**Audio.loadBGNoise(noise)**

#### Changing Background Noise Settings

**# Intended to be used in pre-exercise screen**

**# category: String (Name of Setting, Example: Exercise 1)**

**# setting: String (Setting to be changed, Example: Sound, Volume)**

**# value: String (Value of setting)**

**Database.updateSetting(category, setting, value)**

#### Playing/Stopping Background Noise

**# Play**

**Audio.playBGNoise()**

**# Stop**

**Audio.stopBGNoise()**

# Help Page

This page contains all links and relevant information a user might need when using Auditory Ace as well as an email address to contact with bug reports or help requests.



Help page as of 3/10/24

### Introduction

This section contains the following: a description of the application and its purpose.

“Auditory Ace is a free open source project for users of cochlear implants to work on improving their hearing in an easy and fun-to-use environment.”

### Links to important documentation

This section of the help page will provide links to the GitHub Repository, to this documentation, and to all authors of files that are used with credited permission.

### List of links:

<https://github.com/thorkel2/Auditory-Ace>

<https://www.flaticon.com/free-icons/flash-cards>

### Email Address

[auditoryacereporting@gmail.com](mailto:auditoryacereporting@gmail.com)