CSV File And Word/Sentence List Managers

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