tayLyrics

Author: Jasmine Xu

About

tayLyrics is a lyrics-guessing game that aims to assess how well you - a presumptive Swiftie - know Taylor Swift's lyrics. The app generates a lyric and your goal is to guess the song it comes from.

Your goal is to earn as many points as you can without losing all 5 of your lives, while also making sure not to use too many hints, because those will cost you points. If you are satisfied with your game results, you can add your results to the Leaderboard, where you can see how you match up to other Swifties.

Here is the link to the game. You can explore the associated GitHub repository for further details. Good luck and have fun!

Eventual Aim

Because each artist's songs and lyrics can be obtained programmatically from genius.com, my eventual aim is to provide an interface where fans of any artist can play their favorite artist's lyrics-guessing game. **tayLyrics** is the first of many such games that will be provided. Each game will be customized to be unique to the artist - for example, its name may be a pun related to the artist (e.g. **tayLyrics**), or it may have themes based on colors of the artist's album covers.

All Code

```
import streamlit as st
import pandas as pd
from time import gmtime, strftime
import random
import math
import re
import os
import stringdist
import sqlite3

### CLASSES ###
class Leaderboards:
    """
    A class that manages leaderboards via a SQLite database.

    Args:
        db_path: The path to the database.
    """

    def __init__(self, db_path="leaderboard.db"):

        """
        Constructor
```

```
        """

        self.db_path = db_path
        self.table_names = [f"leaderboard_{diff}" for diff in ["easy",
"medium", "hard"]]
        self.difficulty_mapping = {"Easy (an entire section, e.g. chorus)":
"easy",
                                   "Medium (2 lines)": "medium",
                                   "Hard (1 line)": "hard"}

    def __enter__(self):
        """
        Connects the database and creates the leaderboards for each
        difficulty if they do not already exist.
        """

        self.connection = sqlite3.connect(self.db_path)
        self.cursor = self.connection.cursor()
        self.create_tables()
        return self

    def __exit__(self, exc_type, exc_val, exc_tb):
        """
        Automatically closes the database connection.
        """

        self.connection.close()

    def create_tables(self):
        """
        Method that creates the three leaderboards (one for each
difficulty) if
        they do not already exist.
        """

        for table_name in self.table_names:
            create_query = f"""CREATE TABLE IF NOT EXISTS {table_name} (
                               id INTEGER PRIMARY KEY AUTOINCREMENT,
                               name TEXT NOT NULL,
                               points INTEGER NOT NULL,
                               rounds INTEGER NOT NULL,
                               datetime TEXT NOT NULL)"""
            self.cursor.execute(create_query)
            self.connection.commit()

    def add_to_leaderboard(self, difficulty, results):
        """
        Method that adds a user's results to the corresponding leaderboard.

        Args:
            difficulty: A string specifying the game difficulty; this
                        determines which leaderboard the results are added
to.
```

```
            results: A tuple (name, points, rounds, points, datetime)
containing the user's
            game results.
        """

        db_difficulty = self.difficulty_mapping[difficulty]
        add_query = f"""INSERT INTO leaderboard_{db_difficulty} (name,
points, rounds, datetime)
                        VALUES (?, ?, ?, ?)"""
        self.cursor.execute(add_query, results)
        self.connection.commit()

    def get_leaderboards(self):
        """
        Method that returns the most updated leaderboards, each ranked by
the
        number of points the users have earned.

        Returns:
            A dictionary containing the leaderboards (as pandas dataframes)
            for each difficulty.
        """

        all_leaderboards = {}
        columns = ["ID", "Name", "Points", "Rounds", "Datetime (UTC)"]
        final_cols = ["Rank", "Name", "Points", "Rounds", "Datetime (UTC)"]

        for table_name, long_name in zip(self.table_names,
list(self.difficulty_mapping.keys())):
            self.cursor.execute(f"SELECT * FROM {table_name}")
            rows = self.cursor.fetchall()

            df = pd.DataFrame(rows, columns=columns)
            df["Datetime"] = pd.to_datetime(df["Datetime (UTC)"])
            # the primary determinator of rank is the number of points
earned
            df = df.sort_values(by=["Points", "Datetime (UTC)"], ascending=
[False, True])
            df["Rank"] = df[["Points"]].rank(method="first",
ascending=False).astype(int)
            final_df = df[final_cols].set_index("Rank")
            all_leaderboards[long_name] = final_df

        return all_leaderboards

class Lyrics():
    """
    Class to manage lyric generation.

    Args:
        data: A pandas dataframe containing artist lyrics
    """
    def __init__(self, data):
        """Constructor"""
```

```
        self.data = data
        self.rand_num = None
        self.start_line = None
        self.end_line = None

    def set_random_seed(seed):

        try:
            random.seed(int(seed))
        except ValueError:
            return "Seed must be a valid integer, such as 21 or 2003."

    def generate(self, mode):
        """
        Method that returns the generated lyrics, given a game difficulty.

        Args:
            mode: A string specifying the game difficulty (i.e., whether to
                  generate a whole section, 2 lines, or 1 line)

        Returns:
            An HTML-formatted string containing the generated lyrics.
        """

        self.rand_num = random.randint(0, self.data.shape[0] - 1)

        if mode == "Hard (1 line)":
            self.start_line = self.rand_num
            self.end_line = self.rand_num
            return self.data["lyric"][self.rand_num]

        if mode == "Medium (2 lines)":
            track_name = self.data["track_name"][self.rand_num]
            # check if next line is from same song; it not, give previous
line
            if (self.rand_num <= self.data.shape[0] - 1) and
(self.data["track_name"][self.rand_num + 1] == track_name):
                self.start_line = self.rand_num
                self.end_line = self.rand_num + 1
            else:
                self.start_line = self.rand_num - 1
                self.end_line = self.rand_num

            return "<br>".join(self.data["lyric"]
[self.start_line:self.end_line + 1].tolist())

        else:
            rand_section = self.data["element"][self.rand_num]
            # continue adding lines until section differs
            self.start_line = self.rand_num
            while (self.start_line > 0) and (self.data["element"]
[self.start_line - 1] == rand_section):
                self.start_line -= 1
```

```
            self.end_line = self.rand_num
            while (self.end_line < self.data.shape[0] - 1) and
(self.data["element"][self.end_line + 1] == rand_section):
                self.end_line += 1

            return "<br>".join(self.data["lyric"]
[self.start_line:self.end_line + 1].tolist())

    def get_track_name(self):
        """
        Method that returns the name of the track whose lyrics were
returned.

        Returns:
            The correct track name.
        """
        return self.data["track_name"][self.rand_num]

    def get_album_name(self):
        """
        Method that returns the album of the track whose lyrics were
returned.

        Returns:
            The correct album name.
        """
        return self.data["album_name"][self.rand_num]

    def get_previous_line(self):
        """
        Method that returns the line prior to the generated lyrics.

        Returns:
            The line prior to the generated lyrics; if there is nothing
prior (
            i.e., the generated lyrics was the very beginning of the song),
            "N/A" is returned.
        """
        if (self.start_line > 0) and (self.data["track_name"]
[self.start_line - 1] == self.get_track_name()):
            return self.data["lyric"][self.start_line - 1]
        return "N/A"

    def get_next_line(self):
        """
        Method that returns the line following the generated lyrics.

        Returns:
            The line following the generated lyrics; if there is nothing
            following (i.e., the generated lyrics was the end of the song),
            "N/A" is returned.
        """
        if (self.end_line < self.data.shape[0] - 1) and
```

```
(self.data["track_name"][self.end_line + 1] == self.get_track_name()):
            return self.data["lyric"][self.end_line + 1]
        return "N/A"

    def get_section(self):
        """
        Returns the section of the generated lyrics.

        Returns:
            The section (e.g. Chorus, Verse 1, etc.) of the generated
lyrics.
        """
        return self.data["element"][self.rand_num]

    def get_guess_feedback(self, guess):
        """
        Returns a boolean indicating whether a user guess was correct or
        incorrect. Capitalization is waived, as are minor (within 1/3 of
the
        track name's length) spelling mistakes.

        Returns:
            A boolean; True if the guess is "correct," False otherwise.
        """
        correct_song = self.get_track_name()
        # exceptions where the parentheses should be included in the guess
        if correct_song not in ["Mary's Song (Oh My My My)",
                                "I Can Fix Him (No Really I Can)"]:
            # remove parentheses (e.g. Taylor's Version) from track name
            correct_song = re.sub(r"\([^)]*\)", "", correct_song).strip()
        guess = guess.strip()
        # allowing for minor typos
        track_name_length = len(correct_song)
        allowed_diff = math.ceil(track_name_length * 0.33)
        if stringdist.levenshtein(guess.lower(), correct_song.lower()) <=
allowed_diff:
            return True
        return False

### GLOBAL VARS ###
ALL_LYRICS = pd.read_csv("TAYLOR_LYRICS_JUN2024.csv")
ALL_ALBUMS = ["Taylor Swift",
              "Fearless (Taylor's Version)",
              "Speak Now (Taylor's Version)",
              "Red (Taylor's Version)",
              "1989 (Taylor's Version)",
              "reputation",
              "Lover",
              "folklore",
              "evermore",
              "Midnights",
              "THE TORTURED POETS DEPARTMENT"]
DIFFICULTIES = ["Easy (an entire section, e.g. chorus)",
                "Medium (2 lines)",
```

```
                  "Hard (1 line)"]
GAME_MODES = ["Survival (with 5 lives)",
              "Casual (no lives)"]
POINTS_MAPPING = {"Easy (an entire section, e.g. chorus)": 1,
                  "Medium (2 lines)": 3,
                  "Hard (1 line)": 5}
DIFFICULTY_MAPPING = {"Easy (an entire section, e.g. chorus)": "Easy",
                      "Medium (2 lines)": "Medium",
                      "Hard (1 line)": "Hard"}
ALL_ALBUMS_SHORT = ["Debut",
                    "Fearless",
                    "Speak Now",
                    "Red",
                    "1989",
                    "reputation",
                    "Lover",
                    "folklore",
                    "evermore",
                    "Midnights",
                    "TTPD"]
MODE_MAPPING = {"Survival (with 5 lives)": "Survival",
                "Casual (no lives)": "Casual"}
ALBUMS_MAPPING = {long:short for long, short in zip(ALL_ALBUMS,
ALL_ALBUMS_SHORT)}
ALBUMS_MAPPING_INVERSE = {short:long for long, short in
ALBUMS_MAPPING.items()}
THEME_CSS = {
    "Debut": {
        "background_color": "#006767",
        "button_color": "#597A5D",
        "inputs": "#4A786D",
        "text_color": "white"},
    "Fearless": {
        "background_color": "Tan",
        "button_color": "#BFA684",
        "inputs": "Wheat",
        "text_color": "black"},
    "Speak Now": {
        "background_color": "#7E6480",
        "button_color": "#7B4E7E",
        "inputs": "#9F83A1",
        "text_color": "white"},
    "Red": {
        "background_color": "#712929",
        "button_color": "#5F2020",
        "inputs": "#984747",
        "text_color": "white"},
    "1989": {
        "background_color": "#858EA0",
        "button_color": "#787E89",
        "inputs": "#B1BCD0",
        "text_color": "black"},
    "reputation": {
        "background_color": "#222222",
```

```
        "button_color": "#000000",
        "inputs": "#4C4949",
        "text_color": "white"},
    "Lover": {
        "background_color": "#BB919B",
        "button_color": "#DD8B9F",
        "inputs": "#DBABB7",
        "text_color": "black"},
    "folklore": {
        "background_color": "#7f7f7f",
        "button_color": "#616161",
        "inputs": "#999999",
        "text_color": "black"},
    "evermore": {
        "background_color": "#643325",
        "button_color": "#7f3c10    ",
        "inputs": "#895A38",
        "text_color": "white"},
    "Midnights": {
        "background_color": "#212145",
        "button_color": "#1D1D34",
        "inputs": "#4e4466",
        "text_color": "white"},
    "TTPD": {
        "background_color": "#a79e8f",
        "button_color": "#7D776E",
        "inputs": "#9A9181",
        "text_color": "black"}}

### PAGE CONFIG
st.set_page_config(layout='wide',
                   page_title="tayLyrics",
                   page_icon=":sparkles:",
                   initial_sidebar_state="collapsed",
                   menu_items={'About': "#### tayLyrics: A lyrics guessing
game for Swifties"})

### SESSION STATES ###
if "game_in_progress" not in st.session_state:
    st.session_state.game_in_progress = False
if "lyrics" not in st.session_state:
    st.session_state.lyrics = Lyrics(data=ALL_LYRICS)
if "generated_lyrics" not in st.session_state:
    st.session_state.generated_lyrics = None
if "correct_song" not in st.session_state:
    st.session_state.correct_song = None
if "correct_album" not in st.session_state:
    st.session_state.correct_album = None
if "correct_section" not in st.session_state:
    st.session_state.correct_section = None
if "next_line" not in st.session_state:
    st.session_state.next_line = None
if "prev_line" not in st.session_state:
    st.session_state.prev_line = None
```

```
if "round_count" not in st.session_state:
    st.session_state.round_count = 1
if "guess" not in st.session_state:
    st.session_state.guess = None
if "disable_buttons" not in st.session_state:
    st.session_state.disable_buttons = False
if "correct_feedback" not in st.session_state:
    st.session_state.correct_feedback = ""
if "incorrect_feedback" not in st.session_state:
    st.session_state.incorrect_feedback = ""
if "points" not in st.session_state:
    st.session_state.points = 0
if "round_results" not in st.session_state:
    st.session_state.round_results = []
if "streak" not in st.session_state:
    st.session_state.streak = 0
if "streaks" not in st.session_state:
    st.session_state.streaks = []
if "album_counter" not in st.session_state:
    st.session_state.album_counter = {}
if "difficulty" not in st.session_state:
    st.session_state.difficulty = None
if "game_mode" not in st.session_state:
    st.session_state.game_mode = None
if "albums" not in st.session_state:
    st.session_state.albums = []
if "lives" not in st.session_state:
    st.session_state.lives = 5
if "hints" not in st.session_state:
    st.session_state.hints = 0
if "hints_used" not in st.session_state:
    st.session_state.hints_used = 0
if "gameover_feedback" not in st.session_state:
    st.session_state.gameover_feedback = ""
if "disable_hint_btn" not in st.session_state:
    st.session_state.disable_hint_btn = False
if "hint_feedback" not in st.session_state:
    st.session_state.hint_feedback = ""
if "giveup_feedback" not in st.session_state:
    st.session_state.giveup_feedback = ""
if "album_accs" not in st.session_state:
    st.session_state.album_accs = {}
if "past_game_stats" not in st.session_state:
    st.session_state.past_game_stats = ""
if "enable_leaderboard" not in st.session_state:
    st.session_state.enable_leaderboard = False
if "disable_name_input" not in st.session_state:
    st.session_state.disable_name_input = False
if "submitted_datetime" not in st.session_state:
    st.session_state.submitted_datetime = None
if "rank_msg" not in st.session_state:
    st.session_state.rank_msg = ""
if "start_btn_clicked" not in st.session_state:
    st.session_state.start_btn_clicked = False
```

```
if "hide_buttons" not in st.session_state:
    st.session_state.hide_buttons = False

### FUNCTIONS ###
def apply_theme(selected_theme):
    css = f"""
    <style>
    .stApp > header {{
        background-color: transparent;
    }}
    .stApp {{
        background: {selected_theme['background_color']};
        color: {selected_theme["text_color"]};
        font-family: "Helvetica", "Arial", sans-serif;
    }}
    button[data-baseweb="tab"] {{
        background-color: transparent !important;
    }}
    [data-testid="stSidebar"] {{
        background: {selected_theme['background_color']};
    }}
    button {{
        background-color: {selected_theme['button_color']} !important;
    }}
    button:disabled {{
        background-color: transparent !important;
    }}
    div[data-baseweb="select"] > div, div[data-baseweb="base-input"] >
input {{
        background-color: {selected_theme["inputs"]};
        color: {selected_theme["text_color"]};
        -webkit-text-fill-color: {selected_theme["text_color"]} !important;
        font-weight: 600 !important;
    }}
    p, ul, li {{
        color: {selected_theme["text_color"]};
        font-weight: 600 !important;
        font-size: large !important;
    }}
    h3, h2, h1, strong, .lyrics, h4 {{
        color: {selected_theme["text_color"]};
        font-weight: 900 !important;
    }}
    .lyrics {{
        font-size: 20px;
    }}
    [data-baseweb="tag"] {{
        background: {selected_theme['button_color']} !important;
        color: {selected_theme["text_color"]};
    }}
    th {{
        color: {selected_theme["text_color"]} !important;
        font-weight: 900 !important;
        text-align: left !important;
```

```
        }}
        td {{
            color: {selected_theme["text_color"]} !important;
            font-weight: 600 !important;
        }}
        </style>
        """
        st.markdown(css, unsafe_allow_html=True)

    def clear_guess():
        """Clears the guess text input once an answer is submitted"""
        st.session_state.guess = st.session_state.temp_guess
        st.session_state.temp_guess = ""

    def clear_feedback():
        st.session_state.correct_feedback = ""
        st.session_state.incorrect_feedback = ""
        st.session_state.giveup_feedback = ""
        st.session_state.hint_feedback = ""
        st.session_state.gameover_feedback = ""

    def filter_lyrics():
        """Resets the lyrics generator such that it only generates lyrics from
    the selected albums"""
        selected_albums = [ALBUMS_MAPPING_INVERSE[short] for short in
    st.session_state.albums]
        filtered_lyrics =
    ALL_LYRICS[ALL_LYRICS["album_name"].isin(selected_albums)].reset_index()
        st.session_state.lyrics = Lyrics(data=filtered_lyrics)

    def game_started():

        st.session_state.game_in_progress = True

        clear_feedback()

        if len(st.session_state.albums) != len(ALL_ALBUMS):
            filter_lyrics()

        else:
            st.session_state.lyrics = Lyrics(data=ALL_LYRICS)

        (st.session_state.generated_lyrics,
        st.session_state.correct_song,
        st.session_state.correct_album,
        st.session_state.next_line,
        st.session_state.prev_line,
        st.session_state.correct_section) = regenerate()

        st.session_state.album_counter = {album_name: [] for album_name in
    st.session_state.albums}
        st.session_state.enable_leaderboard = False
        st.session_state.disable_name_input = False
        st.session_state.submitted_datetime = None
```

```
    st.session_state.rank_msg = ""
    st.session_state.round_count = 1
    st.session_state.round_results = []
    st.session_state.disable_buttons = False
    st.session_state.disable_hint_btn = False
    st.session_state.points = 0
    st.session_state.streak = 0
    st.session_state.streaks = []
    st.session_state.hints = 0
    st.session_state.hints_used = 0
    st.session_state.lives = 5
    st.session_state.guess = None
    st.session_state.hide_buttons = False

def new_round():
    st.session_state.round_count += 1
    st.session_state.disable_buttons = False
    st.session_state.disable_hint_btn = False
    st.session_state.hints = 0
    st.session_state.guess = None
    st.session_state.hide_buttons = False

    clear_feedback()

    (st.session_state.generated_lyrics,
     st.session_state.correct_song,
     st.session_state.correct_album,
     st.session_state.next_line,
     st.session_state.prev_line,
     st.session_state.correct_section) = regenerate()

def regenerate():
    generated_lyrics = f'<div class="lyrics">
{st.session_state.lyrics.generate(st.session_state.difficulty)}</div>'
    correct_song = st.session_state.lyrics.get_track_name()
    correct_album = st.session_state.lyrics.get_album_name()
    next_line = st.session_state.lyrics.get_next_line()
    prev_line = st.session_state.lyrics.get_previous_line()
    section = st.session_state.lyrics.get_section()

    return generated_lyrics, correct_song, correct_album, next_line,
prev_line, section

def answered_correctly():
    st.session_state.incorrect_feedback = ""
    st.session_state.points += POINTS_MAPPING[st.session_state.difficulty]
    st.session_state.correct_feedback = f"""That is correct! The answer is
indeed **{st.session_state.correct_song}**,

{st.session_state.correct_section}, from the album **
{st.session_state.correct_album}**.
                                          \n\nYou earned
{POINTS_MAPPING[st.session_state.difficulty]} points and have
                                          {st.session_state.points} total
```

```
points."""
    st.session_state.round_results.append(True)
    st.session_state.guess = None
    st.session_state.disable_buttons = True
    st.session_state.disable_hint_btn = True
    st.session_state.streak += 1
    correct_album_short = ALBUMS_MAPPING[st.session_state.correct_album]
    st.session_state.album_counter[correct_album_short].append(True)
    st.session_state.hide_buttons = True

def answered_incorrectly():
    st.session_state.points -= 1
    st.session_state.incorrect_feedback = f'''"{st.session_state.guess}" is
not correct. Please try again!\n\nYou lost 1 point and have **
{st.session_state.points} total points**.'''
    st.session_state.lives -= 1
    st.session_state.streaks.append(st.session_state.streak)
    st.session_state.streak = 0

    if st.session_state.game_mode == "Survival (with 5 lives)":
        st.session_state.incorrect_feedback += f"\n\nYou lost a life and
have **{st.session_state.lives} lives** left."
        if st.session_state.lives == 0:
            st.session_state.disable_buttons = True
            st.session_state.round_results.append(False)
            correct_album_short =
ALBUMS_MAPPING[st.session_state.correct_album]

st.session_state.album_counter[correct_album_short].append(False)
            st.session_state.gameover_feedback = f'''"
{st.session_state.guess}" is not correct.
                                                \n\n**GAME OVER**: You
ran out of lives! Please start a new game.
                                                \n\nThe correct answer
was **{st.session_state.correct_song}**,
{st.session_state.correct_section}, from the album **
{st.session_state.correct_album}**.'''
            st.session_state.incorrect_feedback = ""
            end_game()
            st.rerun()
    st.session_state.guess = None

def hint():
    st.session_state.hints += 1
    st.session_state.hints_used += 1
    st.session_state.points -= 1

    if st.session_state.hints == 1:
        st.session_state.hint_feedback += f"Hint 1: this song comes from
the album **{st.session_state.correct_album}**"
    if st.session_state.hints == 2:
        st.session_state.hint_feedback += f'\n\nHint 2: the next line of
this song is *"{st.session_state.next_line}"*'
    if st.session_state.hints == 3:
```

```
        st.session_state.disable_hint_btn = True
        st.session_state.hint_feedback += f'\n\nHint 3: the previous line
of this song is *"{st.session_state.prev_line}"*'

def giveup():
    st.session_state.incorrect_feedback = ""
    st.session_state.disable_buttons = True
    st.session_state.disable_hint_btn = True
    st.session_state.points -= 2
    st.session_state.lives -= 1
    st.session_state.round_results.append(False)
    st.session_state.streaks.append(st.session_state.streak)
    st.session_state.streak = 0
    st.session_state.hide_buttons = True
    correct_album_short = ALBUMS_MAPPING[st.session_state.correct_album]
    st.session_state.album_counter[correct_album_short].append(False)

    st.session_state.giveup_feedback = f"""The correct answer was **
{st.session_state.correct_song}**, {st.session_state.correct_section}, from
the album **{st.session_state.correct_album}**\n\nYou lost 2 points and
have **{st.session_state.points} total points**."""

    if st.session_state.game_mode == "Survival (with 5 lives)":
        st.session_state.giveup_feedback += f"\n\nYou lost a life and have
**{st.session_state.lives} lives** left."
        if st.session_state.lives == 0:
            st.session_state.gameover_feedback = f'''**GAME OVER**: You ran
out of lives! Please start a new game.
                                                \n\nThe correct
answer was **{st.session_state.correct_song}**,
{st.session_state.correct_section}, from the album **
{st.session_state.correct_album}**.'''
            st.session_state.giveup_feedback = ""
            end_game()
            return

def end_game():
    # if the game is ended before an answer is provided, count it as
incorrect
    if len(st.session_state.round_results) != st.session_state.round_count:
        st.session_state.round_results.append(False)
        correct_album_short =
ALBUMS_MAPPING[st.session_state.correct_album]
        st.session_state.album_counter[correct_album_short].append(False)

    accuracy_pct = round((sum(st.session_state.round_results) * 100
/st.session_state.round_count), 2)
    possible_pct = round(st.session_state.points * 100 /
(st.session_state.round_count *
POINTS_MAPPING[st.session_state.difficulty]), 2)
    accuracy_str = f"
{sum(st.session_state.round_results)}/{st.session_state.round_count}
({accuracy_pct}%)"
    possible_str = f"
```

```
{st.session_state.points}/{st.session_state.round_count *
POINTS_MAPPING[st.session_state.difficulty]} ({possible_pct}%)"

    accs = {album: (round(sum(ls) * 100/len(ls), 2), sum(ls), len(ls))
            if len(ls) else (0.0, 0, 0) for album, ls in
st.session_state.album_counter.items()}
    st.session_state.album_accs = dict(sorted(accs.items(),
                                       key=lambda x: (x[1][0], x[1][2],
x[1][1]),
                                       reverse=True))
    st.session_state.enable_leaderboard = True if
((st.session_state.game_mode == "Survival (with 5 lives)") and

(len(st.session_state.albums) == len(ALL_ALBUMS)) and

(st.session_state.round_count >= 5)) else False


    st.session_state.past_game_stats = f"""
{DIFFICULTY_MAPPING[st.session_state.difficulty]} difficulty,
{MODE_MAPPING[st.session_state.game_mode]} mode,
{st.session_state.round_count} rounds played
* :dart: Accuracy: {accuracy_str}
* :100: Points out of total possible: {possible_str}
* :bulb: Hints used: {st.session_state.hints_used}
* :fire: Max streak: {max(st.session_state.streaks) if
len(st.session_state.streaks) else 0}
* :moneybag: Total points: {st.session_state.points}
"""

    st.session_state.game_in_progress = False

def name_submitted():

    st.session_state.disable_name_input = True
    st.session_state.submitted_datetime = strftime("%Y-%m-%d %H:%M:%S",
gmtime())

    possible_pct = round(st.session_state.points * 100 /
(st.session_state.round_count *
POINTS_MAPPING[st.session_state.difficulty]), 2)
    possible_str = f"
{st.session_state.points}/{st.session_state.round_count *
POINTS_MAPPING[st.session_state.difficulty]} ({possible_pct}%)"

    game_results = (st.session_state.leaderboard_name,
st.session_state.round_count,
                    possible_str,
                    st.session_state.submitted_datetime,
                    possible_pct)

    # clear the name input text box
    st.session_state.name = st.session_state.leaderboard_name
    st.session_state.leaderboard_name = ""
```

```
    with Leaderboards() as leaderboard:
        leaderboard.add_to_leaderboard(st.session_state.difficulty,
game_results)
        current_leaderboards = leaderboard.get_leaderboards()

    added_to_df = current_leaderboards[st.session_state.difficulty]
    filtered_row = added_to_df[added_to_df["Datetime (UTC)"].astype(str) ==
str(st.session_state.submitted_datetime)]
    added_rank = int(filtered_row.index[0])
    out_of = added_to_df.shape[0]

    st.session_state.rank_msg = f"Your game results were added to the
leaderboard!\nYou ranked in position {added_rank} out of {out_of} total
results."


def highlight_new_row(row):
    """Highlights the row that was just added to the leaderboard in
green"""
    if str(row["Datetime (UTC)"]) ==
str(st.session_state.submitted_datetime):
        return ['background-color: #637C63'] * len(row)
    else:
        return [''] * len(row)

def get_database(path="leaderboard.db"):
    with open(path, "rb") as f:
        return f.read()

### UI ###
with st.sidebar:
    with st.expander(":frame_with_picture: Themes", expanded=True):
        st.radio("Select a theme",
                 options=ALL_ALBUMS_SHORT,
                 index=ALL_ALBUMS_SHORT.index("Midnights"),
                 key="theme")
        apply_theme(THEME_CSS[st.session_state.theme])

    st.divider()
    st.markdown(f"Made with :heart: by Jasmine Xu")
    st.markdown(f"Contact me at <jasminexu@utexas.edu>")

    if os.path.exists("leaderboard.db"):

        st.divider()

        db = get_database()
        st.download_button(label="Download leaderboard",
                           data=db,
                           file_name="leaderboard.db",
                           mime="application/octet-stream")
```

```
buffer1, main_col, buffer2 = st.columns([1, 3, 1])
with main_col:
    st.title("Welcome to :sparkles:tayLyrics:sparkles:!")
    if st.session_state.game_in_progress == False:

        start_tab, past_stats_tab, leaderboard_tab = st.tabs(["Start New
Game", "Past Game Statistics", "Leaderboard"])

        with start_tab:

            with st.expander(":pencil2: Instructions (click to expand)",
expanded=False):
                st.markdown(f"Lyrics range from debut to *
{ALL_ALBUMS[-1]}*. All Taylor's Version vault tracks are included!")
                st.markdown(f"Capitalization and minor spelling errors do
NOT matter!")
                st.markdown("### IMPORTANT GUIDELINES:")
                st.markdown('* Do NOT include "(Taylor\'s Version)" in your
guesses; e.g. "Back to December (Taylor\'s Version)" should simply be "Back
to December."\n* Answer "All Too Well" for BOTH the 5-minute and 10-minute
versions of All Too Well.')

            start_form = st.form("game_settings")
            with start_form:
                st.markdown("### Start a New Game")
                st.selectbox("Select a game difficulty",
                             options=DIFFICULTIES,

index=DIFFICULTIES.index(st.session_state.difficulty) if
st.session_state.difficulty else 0,
                             key="difficulty0")
                st.selectbox("Select a game mode",
                             options=GAME_MODES,

index=GAME_MODES.index(st.session_state.game_mode) if
st.session_state.game_mode else 0,
                             key="game_mode0")
                with st.expander("Advanced options"):
                    st.multiselect("Select albums to generate lyrics from",
                                   options=ALL_ALBUMS_SHORT,
                                   default=st.session_state.albums if
st.session_state.albums else ALL_ALBUMS_SHORT,
                                   key="albums0")
                started = st.form_submit_button(":large_green_square: Start
new game")

            if started:
                st.session_state.difficulty = st.session_state.difficulty0
                st.session_state.game_mode = st.session_state.game_mode0
                st.session_state.albums = st.session_state.albums0

                if len(st.session_state.albums) == 0:
                    start_form.error("Please select at least one album.")
                else:
```

```
                    st.session_state.start_btn_clicked = True

            if st.session_state.gameover_feedback:
                st.error(st.session_state.gameover_feedback, icon="😢")

            # if st.session_state.hint_feedback:
            #     st.info(f"{st.session_state.hint_feedback}", icon="ℹ️")

            # if st.session_state.incorrect_feedback:
            #     st.error(f"{st.session_state.incorrect_feedback}",
icon="🚨")

            # if st.session_state.giveup_feedback:
            #     st.error(f"{st.session_state.giveup_feedback}",
icon="🚨")

            if st.session_state.enable_leaderboard:
                st.info("You can add your scores to the leaderboard!",
icon="ℹ️")

        with past_stats_tab:
            if st.session_state.past_game_stats:
                st.markdown("### Past Game Statistics")
                st.markdown(st.session_state.past_game_stats)
                if st.session_state.album_accs:
                    with st.expander("**Per-album accuracies (click to
expand)**"):
                        s = ""
                        for album_name, tup in
st.session_state.album_accs.items():
                            s += f"* {album_name}: {tup[0]}%
({tup[1]}/{tup[2]})\n"
                        st.markdown(s)
            else:
                st.markdown("#### You must start a game before viewing past
game statistics!")

        with leaderboard_tab:
            st.markdown("### Leaderboards")
            if st.session_state.enable_leaderboard:
                with st.popover(f"Add your results to the leaderboard"):
                    st.markdown("#### Add your results")
                    possible_pct = round(st.session_state.points * 100 /
(st.session_state.round_count *
POINTS_MAPPING[st.session_state.difficulty]), 2)
                    possible_str = f"
{st.session_state.points}/{st.session_state.round_count *
POINTS_MAPPING[st.session_state.difficulty]} ({possible_pct}%)"
                    st.markdown(f"Scores to be added: (Round count
{st.session_state.round_count}, Points of possible {possible_str})")
                    st.text_input("Enter your name",
                                  key="leaderboard_name",

disabled=st.session_state.disable_name_input,
```

```
                                on_change=name_submitted)
                st.markdown(st.session_state.rank_msg)
            else:
                st.markdown(f"Your game results can only be added to the
leaderboard if you played 5+ rounds in Survival mode with all albums
enabled.")

            with Leaderboards() as leaderboard:
                current_leaderboards = leaderboard.get_leaderboards()

            leaderboard_to_show = st.selectbox("Select leaderboard to
display",
                                               options=DIFFICULTIES,

index=DIFFICULTIES.index(st.session_state.difficulty) if
st.session_state.difficulty else 0)
            shown_leaderboard = current_leaderboards[leaderboard_to_show]
            shown_leaderboard =
shown_leaderboard.style.apply(highlight_new_row, axis=1)
            st.markdown(f"### {DIFFICULTY_MAPPING[leaderboard_to_show]}
Leaderboard")
            st.table(shown_leaderboard) #, use_container_width=True)

    if st.session_state.start_btn_clicked:
        game_started()

        st.session_state.start_btn_clicked = False
        st.rerun()

    if st.session_state.game_in_progress == True:
        with st.container(border=True):
            game_tab, stats_tab = st.tabs(["Game", "Current Game
Statistics"])
            with game_tab:
                st.write(f"<h4><u>Round {st.session_state.round_count}<u/>
</h4>", unsafe_allow_html=True)
                st.write(st.session_state.generated_lyrics,
unsafe_allow_html=True)
                st.text("")
                st.text_input("Enter your guess",
                              placeholder="e.g. Back to December or Shake it
Off",
                              key="temp_guess",
                              on_change=clear_guess,
                              disabled=st.session_state.disable_buttons)
                if st.session_state.guess:
                    if
st.session_state.lyrics.get_guess_feedback(st.session_state.guess):
                        answered_correctly()
                    else:
                        answered_incorrectly()

                if not st.session_state.hide_buttons:
                    col1, col2, col3, col4 = st.columns([1.5, 3, 1, 1])
```

```
                    hint_btn = col1.button(":bulb: Hint", on_click=hint,
disabled=st.session_state.disable_hint_btn,
                                           help="You get 3 hints per round;
Hint 1 gives the album, and Hint 2 and 3 give the next and previous lines,
respectively. Each hint deducts 1 point from your total.")
                    giveup_btn = col2.button(":no_entry: Give up",
on_click=giveup,

disabled=st.session_state.disable_buttons,
                                             help="2 points are deducted
from your total if you give up.")

                if st.session_state.hint_feedback:
                    st.info(f"{st.session_state.hint_feedback}", icon="ℹ️")

                if st.session_state.incorrect_feedback:
                    st.error(f"{st.session_state.incorrect_feedback}",
icon="🚨")

                if st.session_state.correct_feedback:
                    st.success(f"{st.session_state.correct_feedback}",
icon="✅")
                    st.button(":arrow_right: Next round",
on_click=new_round)

                if (st.session_state.giveup_feedback) and not
(st.session_state.gameover_feedback):
                    st.error(f"{st.session_state.giveup_feedback}",
icon="🚨")
                    st.button(":arrow_right: Next round",
on_click=new_round)

                col1, col2, col4 = st.columns(3)
                col4.button(":octagonal_sign: End current game",
on_click=end_game, key="end_game")

            with stats_tab:
                st.markdown(f"### In-Game Statistics")
                st.markdown(f"**
{DIFFICULTY_MAPPING[st.session_state.difficulty]} difficulty,
{MODE_MAPPING[st.session_state.game_mode]} mode**")
                st.markdown(f"* :large_green_circle: Round:
{st.session_state.round_count}")

                accuracy_pct = round((sum(st.session_state.round_results) *
100 /st.session_state.round_count), 2)
                possible_pct = round(st.session_state.points * 100 /
(st.session_state.round_count *
POINTS_MAPPING[st.session_state.difficulty]), 2)
                accuracy_str = f"
{sum(st.session_state.round_results)}/{st.session_state.round_count}
({accuracy_pct}%)"
                possible_str = f"
{st.session_state.points}/{st.session_state.round_count *
```

```
POINTS_MAPPING[st.session_state.difficulty]} ({possible_pct}%)"

                st.markdown(f"* :dart: Accuracy: {accuracy_str}")
                st.markdown(f"* :100: Points out of total possible:
{possible_str}")
                st.markdown(f"* :fire: Current streak:
{st.session_state.streak}")
                st.markdown(f"* :moneybag: Total points:
{st.session_state.points}")
                if st.session_state.game_mode == "Survival (with 5 lives)":
                    st.markdown(f"* :space_invader: Lives:
{st.session_state.lives}")
```