SOFTWARE 3 GROUP 1: GUESS THE SONG APP

Group Members
Eliza Younger, Erin Byrne, Natalie Hassall, Sarah Power & Parineeta Pai Galvao De Melo E Souza

OUTLINE

INTRODUCTION

'Guess The Song' is a Python-based musical quiz game that provides an entertaining platform for music lovers to test their knowledge and quick-thinking skills while guessing the artists of popular songs.

Aims & Objectives

- To provide an enjoyable and interactive musical quiz game that challenges players to guess the artists of popular songs.
- Produce a user-friendly and visually appealing app that enhances the gaming experience and encourages frequent play.

Roadmap of the Report

This report will explore the background of the project, including its game logic, Spotify integration, and UI design. It will also cover the specifications, design, and architecture of the system. The implementation details, development approach, team roles, tools used, and challenges faced will be discussed. The report will also elaborate on the testing strategy, evaluation results, and system limitations. Finally, conclusions and future enhancements will be presented.

BACKGROUND

Project Overview

We designed a music-based game designed to challenge players' musical knowledge. Players listen to short audio clips and guess the song artist based on provided multiple choice options.

How To Play

1. The player clicks Start to proceed.
2. The player will be presented with 5 ten second clips from hit songs.
3. Along with each clip will be displayed the song's title.
4. The player must select the correct song artist from a list of four options.
5. A correct answer is worth 10 points. The player's score will be calculated and displayed at the end of the round.

Game Logic

The project integrates with the Spotify API to fetch song data, including titles, artists, and audio preview URLs. This data is used to create game rounds. By integrating with the Spotify API, the game can access real-time data from the UK Top 40 playlist, making it dynamic and relevant. The game provides entertainment and challenges users' knowledge of popular songs and artists.

User Interface Design

The user interface features a start screen, gameplay screen, end screen, and high scores screen. Players initiate the game, listen to song previews, and make selections. The UI design includes imagery and responsive buttons.

GAME SPECIFICATIONS & DESIGN

Functional and Non-Functional Requirements

Functional requirements include the ability to play the game, fetch song data from Spotify, and display results. Non-functional requirements encompass responsive design and smooth audio playback.

Functional Requirements for a minimum viable product

- A single-player game, offering five rounds.
- Ability to start a new game and view high scores at the end of the game.
- Integration with the Spotify API to fetch song data for gameplay.
- User-friendly interface for smooth interaction.
- Random selection of songs from a playlist.
- Audio playback of song previews.
- Display of song title and artist options.
- Scoring system based on correct answers.
- Transition between game rounds and end result display.

Non-Functional Requirements

- Responsive user interface design for different screen sizes.
- Smooth audio playback with minimal delay.
- Secure handling of Spotify API credentials.
- Efficient fetching of song data from the Spotify API.
- Code modularity and maintainability for easy updates.
- Error handling for cases (such as when there is no available song preview).

Design & Architecture

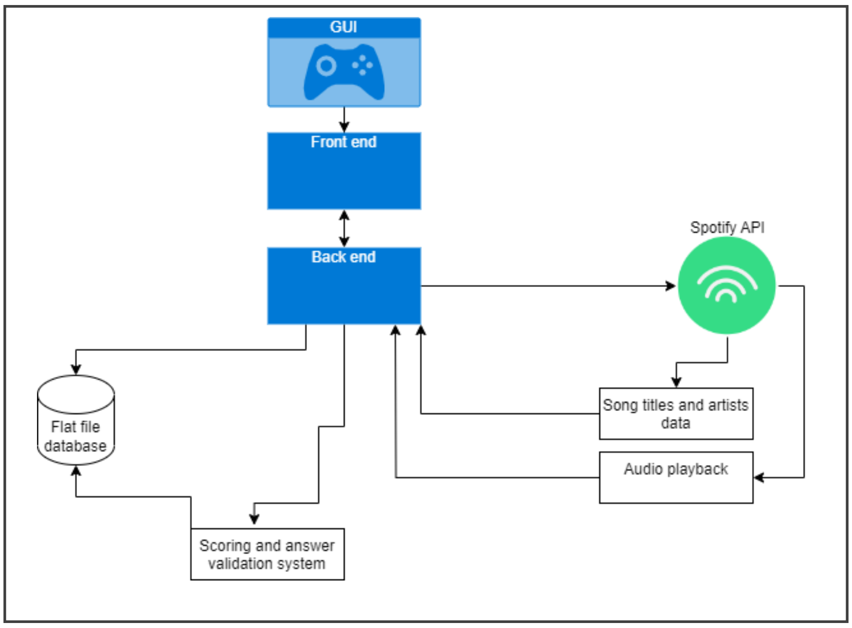

GUI
Front end
Back end
Spotify API
Flat file database
Song titles and artists data
Audio playback
Scoring and answer validation system

Image 1: Figure showing the system architecture of the Guess the Song App

Python Logic Diagram: The Python logic for our diagram of our game is available in Appendix, Image 3.

The architecture follows a Model-View-Controller (MVC) pattern. The Model component represents the application's core logic and data management. It interacts with the Spotify API, fetching data, processing responses, manages the retrieved data. and the game mechanics. The View component represents the user interface and how the data is presented to the user. It includes UI elements and their arrangement. The UI design, layout, and presentation of information to the user remain part of the View component. The Controller component acts as an intermediary between the Model and the View. It handles user input, updates the Model, and updates the View based on changes in the Model. When the user interacts with the UI to trigger API requests (e.g., clicking the button that fetches a new song), the Controller would handle this interaction, trigger the API request in the Model, and update the View accordingly.

Source Tree Directory
We organised our code into separate files in a project to maintain a clean and modular codebase. For our game we separated the code into multiple files based on their responsibilities. A diagram of the Source Tree Directory is available in Appendix, Image 2.

Class Diagram - The class diagram of our game is available in Appendix, Image 4.

IMPLEMENTATION AND EXECUTION:

Development Approach
An iterative and agile development approach was adopted. The team divided tasks based on what element each member was most interested in improving within their skill base. Core tasks were split into segments including Spotify Integration & Game Logic, UI, Database, and Testing. A member took each element within their responsibility to work on and develop, before reviewing progress in regularly scheduled meetings.

Team Member Roles
As a team we decided that we all would take turns being a scrum master, so we each took a week. The Kanban Board and GoogleDocs were created by Natalie. The APIs were researched by Erin and Pari. The Spotify API integration and game logic was completed by Pari. The Figma design layout and HTML were done by Eliza. The final UI design and Kivy integration was carried out by Sarah. The database connections were worked on by Erin and Natalie (please see GitHub for a complete history of contributions).

The API integration and game logic was done by Pari. This included researching the API, reading the documentation, filtering out the required data and writing the first draft of a simple program. Initially the program was designed using Pygame and Tkinter, in which it played a track and asked the user to freely input the artist's name. The second version was designed using Kivy and had 4 quiz-style options for users to choose from. Pari initially refactored the code into smaller files and organised the project's source tree directory.

Natalie and Erin collaborated on the integration of the database. This encompassed two primary tables: one for user authentication and another for user scores. Throughout the process, we made several adjustments based on the progression of the rest of the team's work. We explored various concepts, including the implementation of a username-password combination with hashing, as well as the use of solely usernames, which could be linked to randomly generated ones.

The leadership table, displaying the highest scores, was a feature we had envisioned from the outset. This would showcase the Top 10 scores and lay the groundwork for potential future enhancements, such as assigning varying points based on different game response speeds and difficulty levels. Our initial database design was intricately detailed and ambitious, surpassing our immediate requirements. However, as we delved into coding, we recognised the necessity for a more streamlined iteration.

Eliza created the initial layout prototype using Figma. She then created a webpage using HTML and CSS. Due to our limited knowledge of Flask and integrating Python with HTML, and the time constraint of the project, we decided to switch to using Kivy and make an executable Python program instead. Eliza additionally expressed that she was comfortable with writing unit tests for the program. She also cleared the GitHub main branch and wrote the README.md file.

Sarah took over the UI design and learned how to manipulate and use Kivy. Sarah styled existing code and integrated new Kivy code into the project to improve the appearance and behaviour of the application. She also added the scoreboard/flat use database code as the team was not able to integrate the original databases into the project. Sarah also refactored the game manager code into different python files as the main code grew too large. Finally, she designed and distributed User Acceptance Tests and collated user responses.

Eliza, Pari, Sarah and Natalie wrote unit-tests to ensure proper functionality of the project. Some functions were challenging to test as the integration of the GUI required new approaches of unit-testing which did not necessarily follow the format we had up until this project become familiar (e.g. a function which takes in an input and returns an output). Further unit tests were written by Erin and Natalie but they are not included due to the database not being integrated.

TOOLS AND LIBRARIES USED

We used the following tools to create the project:

- Project deployment and testing: Python and various add-on packages
- Kivy Framework for UI (open-source Python library): Used for creating the user interface and other UI elements. This was used for its scalability to provide a type of user interface for a more enjoyable experience.
- Spotipy Library for Spotify Integration (Python library): Spotify API integration
- Requests: This was used for HTTP requests in Python.
- Datetime: This module was used for handling various time-related operations.
- Random: This Module was used for generating random numbers and performing random selections within Python. We used it to implement a random song generator from the Spotify playlist.
- Unittest with associated modules for mocking and patching (Python's built-in testing framework): It supports test automation, sharing of setup and shutdown code for tests, aggregation of tests into collections, and independence of the tests from the reporting framework. It was used to write unit tests for the different classes and methods.
- Online GoogleDocs survey form was used to collect User Acceptance Tests.

Team collaboration and project storage:

- Git for Version Control – A version control system used to facilitate collaborative work.
- GitHub – A cloud-based version control management system that uses Git to enable collaboration within a development team.
- Kanban Board- Used for task allocation and to schedule future tasks.
- Google Documents - Used for initial project ideas and plans. Used for storing files (such as the project report) to allow the team to access and work on the documents concurrently.
- Zoom – Used to schedule and carry out meetings/debugging sessions.
- Slack – Used for frequent chat communication and impromptu voice calls within the team.

Agile Development Elements:
Incorporating agile principles, we prioritised practices like post-sprint code reviews to ensure code quality and alignment with project goals. Our iterative development approach facilitated ongoing feature enhancement. While these concepts were conceptually strong, our execution experienced some learning curves. This experience has paved the way for future improvements. Despite our regular meetings where we provided detailed updates on progress and achievements, we recognised the need to expedite the integration process. In retrospect, initiating integration earlier would have been advantageous due to its intricacies. This realisation guides our future strategy, aiming for a more streamlined balance between development and integration phases.

IMPLEMENTATION PROCESS AND CHALLENGES

Development progressed in sprints, with weekly meetings to discuss progress and address challenges. As mentioned, each team member assumed responsibility for a distinct aspect of the project, leveraging their individual strengths and areas of expertise. These meetings served as platforms for team members to report on their individual contributions, seek feedback, and discuss potential roadblocks. This approach not only promoted accountability but also enabled the team to stay aligned with the project's goals and milestones.

Achievements

- Successfully implemented the core game logic, including random song selection, playback, and answer evaluation.
- Efficiently integrated with the Spotify API to fetch song data.
- Developed an engaging user interface using the Kivy framework.
- Divided development tasks among team members based on expertise.
- Embraced an agile approach with iterative development and code reviews.
- Enhanced knowledge of Python, UI design, and API integration.
- Successfully refactored and optimised code for improved readability and maintainability.

Challenges

- Connecting to the Spotify API took considerable research and testing. Two other APIs were considered but Spotify was found to be the best.
- Achieving proper audio fetching and synchronisation took numerous attempts.
- Building the main game logic using Kivy and classes was a challenge as Kivy was a new framework to learn, which consumed a lot of project time. Due to this the UI design also took a lot of time and effort.
- Refactoring the main file structure to adhere to best practices and testing principles was challenging as we encountered import issues for people using computers with different Operating Systems or versions of Python.
- Writing appropriate unit tests for functions with multiple dependencies was challenging.
- Rapid refactoring of the main file structure close to the deadline demanded quick action.

Adjustments to project proposal
At the beginning of the project, the team began to develop the project's user interface with the idea that it would be web-based. Initially, the decision was made to create the interface using HTML and CSS, however, during the early stages of development, the team recognised an alternative method for more seamless integration between the interface and the Python components. Consequently, a strategic decision was made to transition to using Kivy, a Python framework for building cross-platform applications.

Implementation of the ‘Tkinter’ module for UI enhancement was switched to Kivy as Kivy had a more modern user interface. The original idea to store data with SQL changed due to problems with integration and time constraints. The decision to refactor was taken to improve code maintainability, testability, and readability. The choice to prioritise refactoring over implementing complex features was based on available time and skill level. There were also some unresolved issues where the same song could play twice as the random song selection was not unique. Also, sometimes the artist's name appeared twice if they were associated with multiple songs in the playlist.

TESTING & EVALUATION

TESTING STRATEGY

The testing phase aimed to ensure the functionality, reliability, and performance of the app's system. Key objectives included Quality Control, validating user interactions and identifying and fixing potential bugs. Quality Assurance was achieved by using OOP principles while organising our source tree directory and while writing our code.

Our testing strategy consisted of a combination of User Acceptance Testing (UAT) and Unit Testing. The tight integration of the logic with the UI components made isolating units for traditional tests difficult, and although unit tests were written where possible, equal weight was given to validating user experience through UAT. The design of the software itself limited user input (as options were multiple choice and there were no text input fields) reducing the possibility of edge cases.

TESTING METHODS

User Acceptance Testing: (UAT) is a phase of software testing where the software is tested by end-users to determine whether it meets the specified requirements and is ready for 'release'. The primary goal of UAT is to ensure that the software meets the needs of its intended users and that it operates as expected in a real-world environment.

Unit Testing and Integration Testing

- Unit Testing: Individual components of the system were tested in isolation to verify their correctness and expected behaviour. The input-process-output format was not always the *de facto* function state, and the unit-tests were adapted to include void function testing and mocking.

- Integration Testing: Our main goal was to maintain a seamless flow of data between different parts of the project. To achieve this, we adopted a step-by-step approach to integration testing. We examined how each new component interacted with the existing project components, ensuring they worked well together. This approach helped us ensure that all the components fit together smoothly and were compatible. Nonetheless, it's worth noting that this process wasn't without its difficulties. We faced technical challenges that required careful attention and hard work to overcome, in order to successfully integrate all the components.

- Regression Testing: After each code change, we performed a manual regression testing for the game. Since this game involves user interactions and game mechanics, manual testing would focus on the user interface, gameplay, and overall functionality. This involved systematically testing various functionalities of the game to ensure that recent code changes, updates, or new features have not introduced any regressions or unintended issues. And when this occurred, we spent time correcting the newly added code or functionality to correct the program. For example, in the second version of the game, when the multiple-choice answer options were added, we noticed that the game would run through tracks in seconds. Here we had to stop and correct our code before proceeding to the next stage.

UAT Test Cases and Scenarios:

A demonstration version of the software was distributed to 14 respondents, alongside a UAT template via Google Forms. The UAT template covered the expected functionality under the four main testing scenarios (Starting the game, Gameplay, End Screen and High Score Screen). The majority of testers experienced a smooth start to the game, with title screens and buttons working effectively. Gameplay interactions, including song previews, correct/incorrect answer selections, and next question loading, were reported as smooth and engaging.

Testers found the end screen's "Game Over" label and final score presentation to be clear and user-friendly. Play Again and View High Scores buttons were functional, and the ability to end the game was straightforward. Testers reported enjoying the game, highlighting its engaging nature, smooth transitions, and responsive button feedback.

Some feedback included suggestions to enhance visual aesthetics and provide clearer instructions for new users. Minor issues reported included audio playback synchronisation issues on some devices, and a limited number of songs available for gameplay (which could be fixed by using a different Spotify playlist).

CONCLUSION:

The Guess The Song App project achieved its objectives of creating an interactive music-based game. The team successfully integrated with the Spotify API, implemented game mechanics, and designed an engaging user interface.

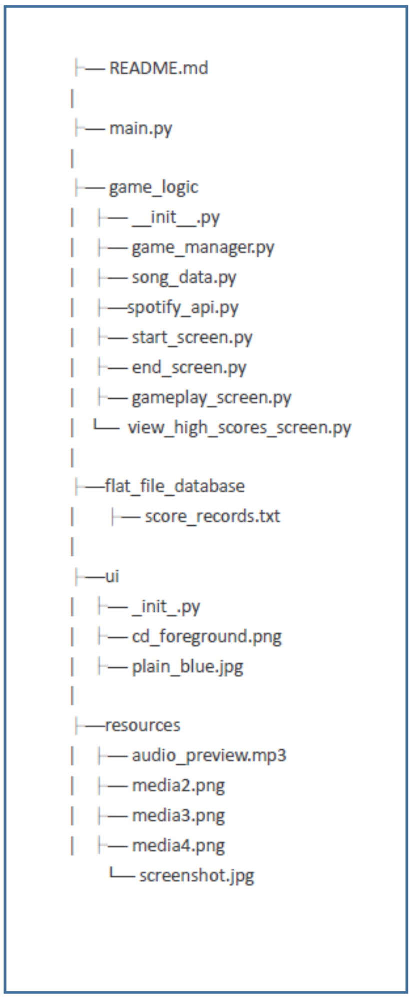
ACHIEVEMENTS AND LESSONS LEARNED:

The Guess The Song App project successfully created an engaging game. The team learned about API integration, UI/UX design, and agile development practices.

FUTURE DIRECTION

- Improved audio playback synchronisation
- Expanding the song pool to include different genres, eras and playlists
- Multiplayer mode
- Difficulty levels to include the speed of responses, reduction in length of time the song is played for and song pool
- Gameplay Engine to transition between “Name that Tune” (a version of the game where the question is reversed so that the song name is guessed rather than the artist) if we expand to enabling short clips/song phrases as part of the game.

Appendix



```
├── README.md
|
├── main.py
|
├── game_logic
|   ├── __init__.py
|   ├── game_manager.py
|   ├── song_data.py
|   ├──spotify_api.py
|   ├── start_screen.py
|   ├── end_screen.py
|   ├── gameplay_screen.py
|   └── view_high_scores_screen.py
|
├──flat_file_database
|     ├── score_records.txt
|
├──ui
|   ├── _init_.py
|   ├── cd_foreground.png
|   ├── plain_blue.jpg
|
├──resources
|   ├── audio_preview.mp3
|   ├── media2.png
|   ├── media3.png
|   ├── media4.png
|     └── screenshot.jpg
```

Image 2: Figure showing a diagram of the Source Tree Directory

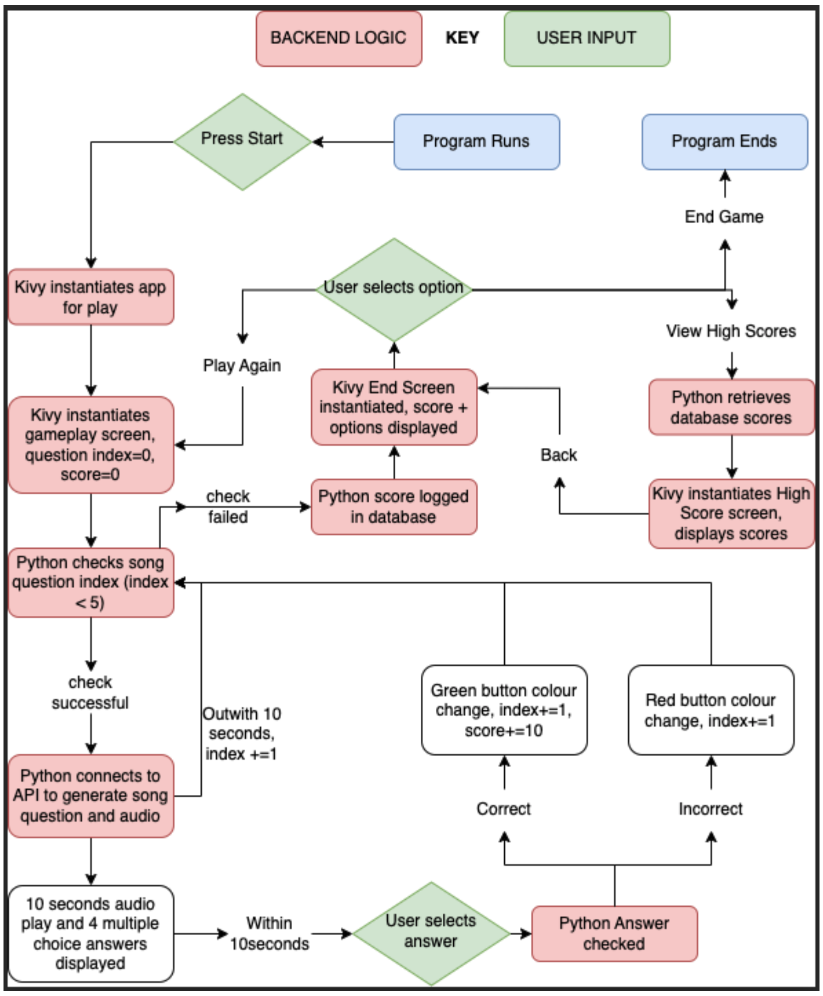

BACKEND LOGIC
KEY
USER INPUT
Press Start
Program Runs
Program Ends
End Game
Kivy instantiates app for play
User selects option
View High Scores
Play Again
Kivy End Screen instantiated, score + options displayed
Python retrieves database scores
Kivy instantiates gameplay screen, question index=0, score=0
Back
check failed
Python score logged in database
Kivy instantiates High Score screen, displays scores
Python checks song question index (index < 5)
check successful
Outwith 10 seconds, index +=1
Green button colour change, index+=1, score+=10
Red button colour change, index+=1
Correct
Incorrect
Python connects to API to generate song question and audio
10 seconds audio play and 4 multiple choice answers displayed
Within 10seconds
User selects answer
Python Answer checked

Image 3: Python logic of the program

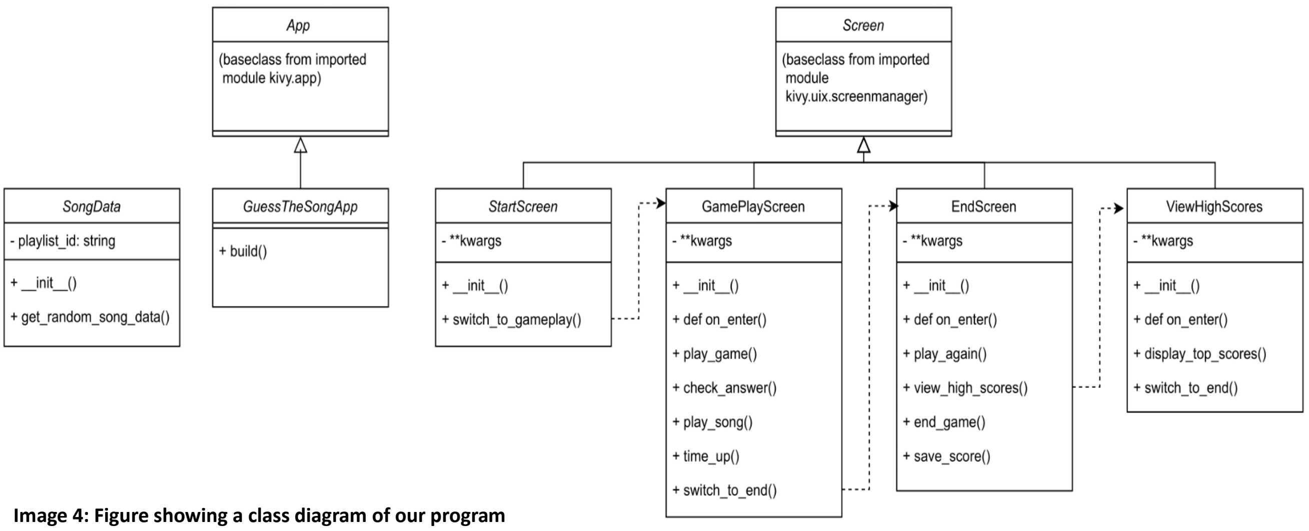
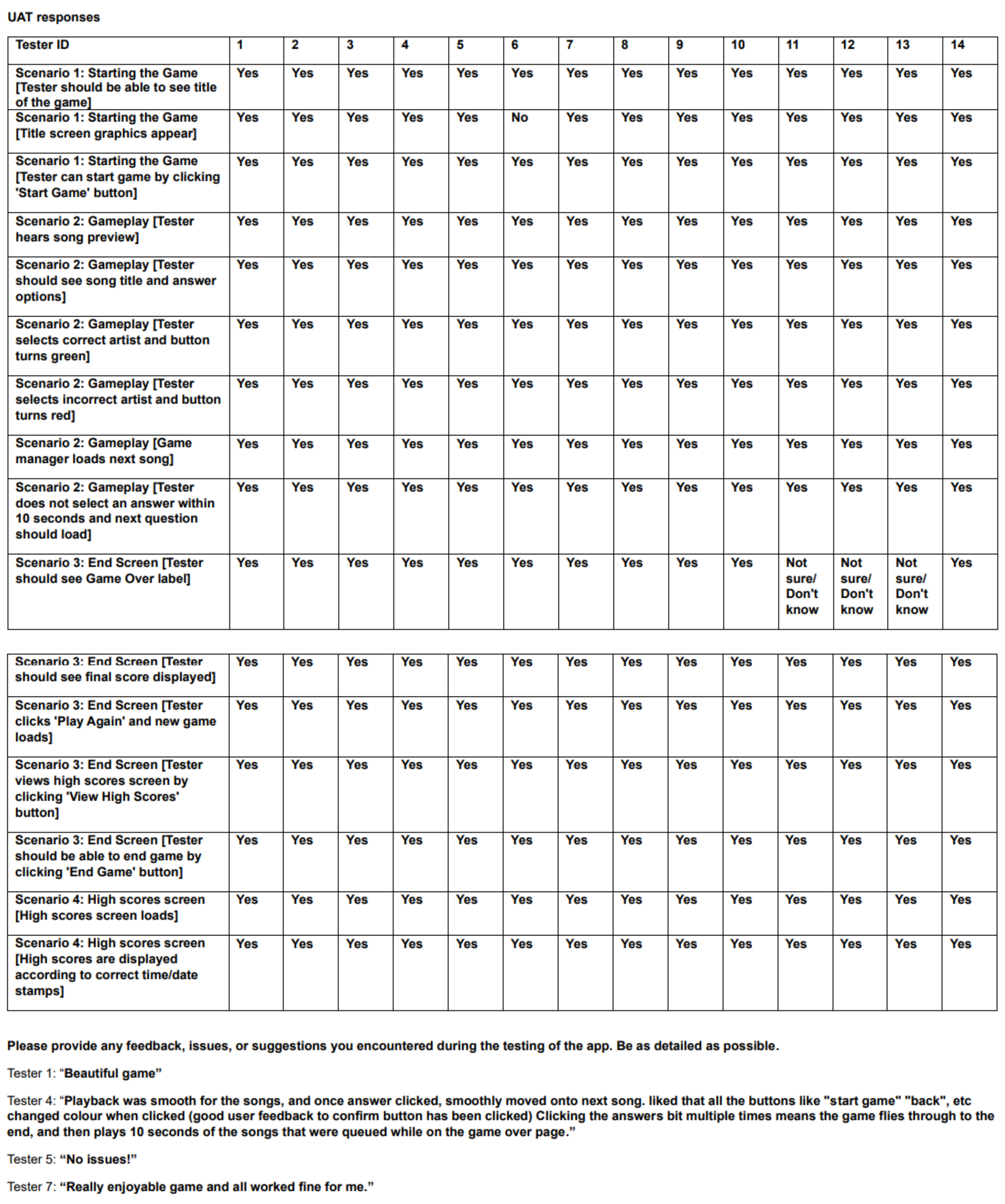


Image 4: Figure showing a class diagram of our program

Appendix continued.



UAT responses

Tester ID	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Scenario 1: Starting the Game [Tester should be able to see title of the game]	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Scenario 1: Starting the Game [Title screen graphics appear]	Yes	Yes	Yes	Yes	Yes	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Scenario 1: Starting the Game [Tester can start game by clicking 'Start Game' button]	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Scenario 2: Gameplay [Tester hears song preview]	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Scenario 2: Gameplay [Tester should see song title and answer options]	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Scenario 2: Gameplay [Tester selects correct artist and button turns green]	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Scenario 2: Gameplay [Tester selects incorrect artist and button turns red]	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Scenario 2: Gameplay [Game manager loads next song]	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Scenario 2: Gameplay [Tester does not select an answer within 10 seconds and next question should load]	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Scenario 3: End Screen [Tester should see Game Over label]	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Not sure/ Don't know	Not sure/ Don't know	Not sure/ Don't know	Yes
Scenario 3: End Screen [Tester should see final score displayed]	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Scenario 3: End Screen [Tester clicks 'Play Again' and new game loads]	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Scenario 3: End Screen [Tester views high scores screen by clicking 'View High Scores' button]	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Scenario 3: End Screen [Tester should be able to end game by clicking 'End Game' button]	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Scenario 4: High scores screen [High scores screen loads]	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Scenario 4: High scores screen [High scores are displayed according to correct time/date stamps]	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes

Please provide any feedback, issues, or suggestions you encountered during the testing of the app. Be as detailed as possible.

Tester 1: **"Beautiful game"**

Tester 4: **"Playback was smooth for the songs, and once answer clicked, smoothly moved onto next song. liked that all the buttons like "start game" "back", etc changed colour when clicked (good user feedback to confirm button has been clicked) Clicking the answers bit multiple times means the game flies through to the end, and then plays 10 seconds of the songs that were queued while on the game over page."**

Tester 5: **"No issues!"**

Tester 7: **"Really enjoyable game and all worked fine for me."**

Image 5: UAT Responses and Feedback