Technical Report

Designed by: Lizao Wang, Annie Liu, Vera Ye

I. Abstract Data Types

1. A **Linked Binary Tree** is used to represent a number of questions that the user would be prompted to answer. The user would select their desirable answer from two possible choices, either A or B, and continue to do so until no more questions show up. Answering these choice questions would effectively guides the user's path from the root of this binary tree to the node which contains the corresponding matching song for the user.
2. An **Array Queue** is used to hold information about the encrypted names of the songs in the database. Each queue contains two different encrypted songName Strings and one imageFile String which is a string representation of the name of the image file. The encrypted songName Strings are generated through two different encrypt() methods with different level of difficulty and enqueued in this decreasing level of difficulty, so as to offer hints for the user to guess the name of the song. The hints would be given out in hard → easy order to challenge the user to make the best guesses, so each string in the queue would be dequeued as soon as the user makes one wrong guess.
    - If the user has made the right guess, the corresponding CD cover image of that song would be displayed.
    - If the user could not successfully guess the song's name after three rounds, the song name would be revealed and the image would be displayed.

II. Classes

1. **musicBinaryTree**: contains Linked Binary Tree. It helps to make a prediction of which piece of music fits you the best. For each question, the left child (A) represents one answer and the right child (B) represents another answer. Begin with the question at the root, following the appropriate path until a leaf node (which contains the piece of music for the user) is reached.
2. **musicWizardPanel:** contains following GUI content:

a. **questionPage**: when users click play, the program will call this class. It will print each question and display two radio buttons under it. Each radio button is associated with an answer. There will be a quit button on the top right corner. If users click quit button, the program will exit.

b. **landingPage**: when the game is initialized, prints the name of this project on the screen, the statement "We know your favorite piece" under it, as well as the play button. Set text font, color, and background.

c. **resultPage**: after users finish answering questions, there will be a play button to play the music, a stop button to stop the music, a text field for users to enter their guesses, and a button to check the answer.

d. **finalPage**; displays the image of this music by getting the name of the image from encryptedNamesQueue. There will be a play button to play the music, a stop button to stop the music, a quit button to exit the game, and a button to replay the game.

3. **musicWizardGUI:** contains a main class and calls musicWizardPanel**.**
4. **usersGuess**: a helper class that implements the Comparable interface. It will import Scanner to scan user input. It will contain methods to play/stop the music, and to compare user input with the name of the music. If they are the same, finalPage will be displayed. Otherwise, it will get the encrypted strings from encryptedNamesQueue. Print the hint and ask user to input another guess. After three rounds of guessing, finalPage will be automatically displayed.
5. **encryptedNamesQueue**: contains a queue of encrypted names of the songs. There is an array queue for each song. Each queue will contain 2 encrypted Strings and one String (name of the image).

III. Actions

1) **musicBinaryTree**

a) **choose():** prompts the user to answer each of the questions by choosing between choice A and choice B, and continues until all questions have been asked and all corresponding answers have been stored. Then determines the matching song for the user.

b) **musicBinaryTree():** the constructor that initializes the Linked Binary Tree with fixed questions stored inside as nodes of the tree.

2) **musicWizardPanel**
    a) **musicWizardPanel():** the constructor that initializes JRadioButton, JButtons, JTextField, JApplets, and JLabels, and sets text font, size, and background color.
    b) **actionPerformed()**:
        i) **questionPage**
            (1) When one of the radio button is pressed, it will print the the next question following this answer.
            (2) When the "Quit" button is pressed, the game will exit.
        ii) **landingPage**
            (1) When the "Play Game" button is pressed, begin playing the game.
        iii) **resultPage**
            (1) When the "Play" button is pressed, play the music.
            (2) When the "Check Answer" button is pressed, if the user input is the same as the name of the song, call finalPage. Otherwise, continue dequeuing the encrypted Strings from encryptedNamesQueue;
            (3) When the "Stop" button is pressed, stop the music.
            (4) When the "Quit" button is pressed, exit the game.
        iv) **finalPage**
            (1) When the "Stop the Game" button is pressed, stop playing the current selection.
            (2) When the "Replay the Game" button is pressed, stop playing the current selection, clear everything and restart the game.
            (3) When the "Quit" button is pressed, stop playing the current selection and exit the game.

3) **usersGuess**
    a) **readInput():** scans user input and saves it as an input String. Updates the input String every time there's a new user input.

b) **compareTo()**: compares two Strings: the user input String and songName String. It returns:
    i) 0 if two Strings are the same.
    ii) positive/negative numbers if the two Strings are not the same.

4) **encryptedNamesQueue**
    a) **hardEncrypt():** encrypts each songName String by using a password String to shift each character in the string correspondingly, based on the characters in the password String, and enqueues them into their respective encryptedNamesQueue.
    For example, if the original songName String is “havana”, and the password String is “CAT”, the shifts would not be the same for each character, but are:
        i) +2 for the first character (since C is in position 2 of the alphabet);
        ii) +0 for the second character (since A is in position 0 of the alphabet);
        iii) +19 for the third character (since T is in position 19 of the alphabet), and then again:
        iv) +2 for the fourth;
        v) +0 for the fifth;
        vi) +19 for the sixth, and so continues up until the last character of the songName String.

    So, the returned encrypted String would be “jaocnt”.
    b) **easyEncrypt():** encrypts each songName String by flipping each string, and enqueues them into their respective encryptedNamesQueue.
    For example, if the original songName String is “havana”, then the returned encrypted String would be “anavah”.