Technical Report

Designed by: Lizao Wang, Yaxin Liu, Vera Ye

I. Abstract Data Types

1. A **Linked Ternary 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 three possible choices, A, B, or C, 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 ternary 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. 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 makes the correct guess at any point, the song name would be revealed.
    - If the user could not successfully guess the song's name after three rounds, the song name would be automatically revealed.

II. Classes

1. **musicTernaryTree:** contains LinkedTernaryTree. Begin with the question at the root, the user will follow the appropriate path until a leaf node (which contains the piece of music for the user) is reached.
2. **musicWizardPanel:** represents an expert system that prompts the user to choose answers from an underlying LinkedTernaryTree. It asks him/her to make guesses about the name of the song and allows the user to play/stop that piece of music. If the user guesses the correct song name at any point, the panel will go to the final page, showing the user the song name and its singer. Otherwise,

after three unsuccessful guesses, the panel will show the final page. It contains following GUI interfaces:

   a. **landingPage**: when the game is initialized, display the name of this program, Music Wizard, on the screen and a 'Play Game' button. Set text font, color, and background.
   b. **questionPage(s)**: when the user clicks 'Play Game', the program will transfer to this page. It will print each question in turn and display three radio buttons under it. Each radio button is associated with an answer that will lead the user to another question. There will be a quit button at the top.
   c. **guessingPage(s)**: after the user finish answering all the questions, a song will be generated for him/her in the system, but it will not be revealed to the user. There will be a play button to play the music, a stop button to stop the music, a text field for users to enter guesses, and a button to check the answer.
   d. **finalPage**: display the name of the song and its artist. There will be a play button to play the music, a stop button to stop the music, and a quit button to exit the game.

3. **musicWizardGUI:** contains a main class and calls the musicWizardPanel**.**
4. **usersGuess**: a helper class that implements the Comparable interface. It will contain methods to compare user input with the name of the song. If they are the same, finalPage will be directly displayed. Otherwise, it will get the encrypted strings from encryptedNamesQueue, print the hints and ask user to input another guess. After three rounds of guessing, finalPage will be automatically displayed.
5. **encryptedNamesQueue**: contains a hardEncrypt method that implements the Vigenere encryption, an easyEncrypt method that reverse the input, and a buildArray method that builds an array queue and enqueues the two corresponding encrypted messages into the queue. It returns a queue of encrypted names of the songs. There is an array queue for each song.
6. **songNamesVector**: contains instance data including the arrays of songs that each corresponds to a leaf node in the LinkedTernaryTree. When the user has reached a leaf node, the corresponding array would be recognized and one song inside the array would be randomly generated.

III. Actions

1) **musicTernaryTree()**
    a) **musicTernaryTree():** contains the questions and choices used in the program.
    b) **getMusicTree():** a getter that returns the musicTree.

2) **musicWizardPanel**
    a) **musicWizardPanel():** contains instance data including JButtons, JRadioButtons, ImageIcons, JLabels, JTextField, JPanels, and AudioClips as GUI interactive components in the program. Set default layout, color, and dimension. Initializes the landing page.
    b) **ActionListenerOne**
        i) When playGame button is pressed, it will switch from landing page to initial question page.
            (1) Initializes quitButton, questionLabel, buttonA, buttonB, and buttonC.
            (2) Creates and initializes a new questionPanel which contains quitButton, questionLabel, buttonA, buttonB, and buttonC.
        ii) **ActionListenerTwo**
            (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.
            (3) When the user has reached the leaf node of the tree, it will switch from questionPages to guessPages.
                (a) It will get a randomly generated song from songNamesVector class based on user's path in musicTree.
                (b) Set buttonA, buttonB, buttonC to invisible and 'disable' them.
                (c) Initializes guessLabel, userInput, playMusicButton, stopMusicButton, and checkAnswerButton.

(d) Creates and initializes a new guessPanel which contains guessLabel and userInput.

(e) Creates and initializes a new controlPanel which contains playMusicButton, stopMusicButton, and checkAnswerButton.

(f) Add controlPanel to questionPanel. Add guessPanel to questionPanel.

(4) Import and set up music url files.

iii) **ActionListenerThree**

(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, display 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.

3) **usersGuess**

a) **getGuess():** a getter that returns String guess (user's guess).

b) **compareTo()**: compares the original songName String to the user input (guessed) 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) **findShift():** Helper method. Finds the index of characters in password String, stores them in an integer array and returns it.

b) **hardEncrypt():** encrypts each songName String by using the password String, "CSROCKS", and correspondingly shifting each character in the String. Returns the encrypted char [ ].

c) **easyEncrypt():** encrypts each songName String by flipping each String backward. Returns the encrypted char [ ].

d) **toString()**: Helper method. Turns a char [ ] into a String and returns it.

e) **buildArray()**: Build a corresponding ArrayQueue that contains the two encrypted Strings. Returns ArrayQueue<String> that contains the two encrypted results.

5) songNamesVector

a) **songNamesVector():** creates and initializes 27 String arrays. Each array contains three songs. Add these arrays into a Vector of String array.

b) **getOneSong()**: Returns one randomly generated song from the corresponding array given its index in the Vector.