HANGMAN PROJECT: PART 1

Due November 7, 2019

Mike Kennedy - 108715992
Stony Brook University
ESE 224: Computer Techniques for Electronic Design II

Contents

1 Summary

1.1 Structure

The game is run by the file play_game.cpp. This file is responsible for the interaction between the user and the underlying logic of the game. This file calls hangman.h which subsequently calls game_board.h. Sequentially, the program runs as follows:

1. play_game.cpp creates a game object of type hangman
2. hangman.h creates an object of type game_board
3. The user is continuously polled for input by play_game.cpp
   (a) play_game.cpp validates the user input and passes it into the hangman object
   (b) hangman processes the guess and uses this data to update the game board
   (c) hangman checks whether the game has been won or lost, and terminates if applicable
4. play_game.cpp displays the correct phrase at the end of the game

1.2 Test cases

The application is capable of handling any input type from the user and will alert them if their input is invalid. It also can handle phrases containing spaces and return characters without error.

2 play_game.cpp

This is the file responsible for running the game. It contains two methods - removeReturn and validInput, as well as the main method. This file is responsible for interacting with the user. It validates the input format, and handles exceptions thrown by the hangman and board classes that are implemented. After creating a game, it continuously polls the user until the game is completed, whether won or lost, then terminates.

In order to accomplish this, it first loads a word from **WordList.txt**, which must be in the same directory as the run files. Following the game creation, the main method calls removeReturn and passes the phrase into the game header, hangman.h. The user is then prompted to guess a letter, which, after validation by the validInput method, is also passed through hangman.h and into the game_board.h header to display the result.

2.1 removeReturn

When importing the phrase from the text file, the word or phrase has a return character - \r - attached at the end. The *removeReturn* method takes the phrase as a parameter and returns a string with any return characters removed. This is necessary to avoid requiring having the software interpret the return character as a part of the phrase to be guessed.

2.2 validInput

Once the game begins, the user is continuously polled for input until the game ends. Each input is validated using validInput. This method takes the user input as a string parameter and returns a Boolean - *true* if the input is acceptable or *false* otherwise. validInput checks first the length of the user input. If the input is not a single character, the method immediately returns *false*. Otherwise, the method searches a global string of invalid input characters for the character input by the user. It returns whether or not the character is found in the list of invalid characters.

3 hangman.h

This header file operates game play. It takes the user input and manages the communication between the game board.

3.1 Variables

This file contains six individual private variables that are essential to the operation of the game. The first is **MAX_FAILS**. This is an integer value that tells the software the maximum allowable incorrect guesses before the game is lost. **board** is a variable of type game_board that represents the board on which the game is being played. The string variables **phrase** and **pastGuesses** contain the target phrase to be guessed by the user, and a string containing the characters that were already guessed, respectively. The final two private variables are integers. **incorrectGuesses** holds the value of the number of times the user has guessed incorrectly. **numSpaces** contains the number of spaces present int the target phrase. This is vital for both displaying the phrase correctly and calculating how many letters are remaining in the puzzle.

3.2 Private methods

3.2.1 containsLetter

This method takes two parameters, the string to be searched and the character to search for within that string. It then returns a Boolean value indicating whether or not the letter was found within the string.

3.2.2 lettersRemaining

lettersRemaining is a method that takes no parameters and returns an integer. It loops through the target phrase and compares each character to the list of past guesses. It returns the number of characters in the phrase that have not yet been guessed.

3.3 Public methods

3.3.1 Constructors

There are two constructors for the hangman class. The default constructor sets the target phrase to *"default phrase"*. Alternatively, the other constructor takes a string as a parameter and sets this as the target phrase. Each constructor upon declaring the phrase passes this phrase to board and displays it to the user.

3.3.2 getPhrase

This method is a typical getter. It returns the target phrase to the user.

3.3.3 guessLetter

guessLetter takes a character as a parameter and treats that character as a guess towards the target phrase. If the character has already been guessed, the method will throw an error (see Section 5). If the character has not yet been guessed, it is then added to the string of past guesses. Finally, the method checks whether the guess is correct by calling containsLetter and updating the board display.

3.3.4 gameWon

This is a Boolean method that returns the result of the comparison between lettersRemaining and 0. A zero value for lettersRemaining indicates the user has correctly guessed all of the characters in the phrase.

3.3.5 gameLost

gameLost preforms the inverse operation of gameWon. This method compares the number of incorrect guesses by the user to the maximum allowable (**MAX_FAILS**). If the user has reached the maximum allowable number of failed guesses, the method returns true indicating the game has been lost.

3.3.6 guessesRemaining

This method returns an integer representing the number of guesses the user is still allowed to make.

4 game_board.h

4.1 Variables

This class implements only two private variables. The integer **numWrong** represents the number of incorrect guesses that have taken place at the current point in the game. This variable value influences the display. The other variable is the string, **phrase**. This variable stores the desired phrase and is also used in display. The number of characters both guessed and not must be displayed by this class appropriately.

4.2 Private methods

4.2.1 hangedMan

This method takes the past guesses as a parameter and returns a string. The string returned is generated by comparing the guesses to the target phrase. The number of incorrect guesses indicates how the image is to be displayed.

4.2.2 formattedPhrase

Similarly to the hangedMan method, this method takes the past guesses as a parameter and returns a string. The string returned is the target phrase with the correct guesses displayed, and the characters that have not yet been guessed represented by an underscore ('_').

4.2.3 wrongGuesses

This method has two functions. It takes as a parameter the list of past guesses and returns a string containing the characters from within the past guesses that were incorrect. It also uses the length of this string to set the value of **numWrong**.

4.3 Public methods

4.3.1 Constructors

The default constructor establishes a board in which the phrase is an empty string. There exists a second constructor that takes as a parameter a string which is set to the target phrase.

4.3.2 setPhrase

This method is a typical setter. It has no return value, but takes a string as a parameter and uses it to set the target phrase.

4.3.3 displayBoard

displayBoard is called to display the current board setup. This method calls three methods. It calls wrongGuesses in order to display the past incorrect guesses, hangedMan, which shows the current state of the game, and finally formattedPhrase to display the current state of the puzzle.

5 hangman_exceptions.h

This header contains only the exception that is to be called by hangman.h when the user attempts to guess the same letter twice.