Rushi Samirbhai Patel RS397441@dal.ca
B00886157

Assignment 4

Boggle Puzzle Game

Test Cases:

Input Validations

1. boolean getDictionary()
    - Stream is null.
    - Stream has one character word.
    - Stream has two words in the same line separated by whitespace.

2. boolean getPuzzle()
    - Stream is null.
    - Stream has rows of uneven length of letters (error condition).

3. List<String> solve()
    - The puzzle is solved before generating the puzzle (when the puzzle is not ready).

Boundary Cases

1. boolean getDictionary()
    - Stream has a combination of uppercase and lowercase characters.
    - Stream has two character words.
    - Stream has duplicate words.
    - Stream contains a combination of words(with characters more than one) and a character as word.
    - Stream has an apostrophe in some words.
    - Stream has all the same words.

2. boolean getPuzzle()
    - Stream has only one character.
    - Stream has no characters.
    - Stream has two characters.
    - Stream has an apostrophe as one character.
    - Stream has all the same characters.

3. List<String> solve()
    - The puzzle is solved before generating the dictionary.

- The puzzle is solved when it is empty (has no characters).
- The puzzle is solved when it has only one character.
- The puzzle is solved when the dictionary has no words.

4. String print()
    - The puzzle is printed before generating the dictionary.
    - The puzzle is printed when it is empty.
    - The puzzle is printed when it has only one character.
    - The puzzle is printed when the puzzle is not generated.

Control Flow Cases

1. boolean getDictionary()
    - Stream has one word.
    - Stream has more than one word.
    - Stream contains words on different lines.

2. boolean getPuzzle()
    - Stream has more than two characters.
    - Stream has the same number of characters in each row of the puzzle.

3. List<String> solve()
    - Solve the puzzle when the puzzle is ready.
    - Solve the puzzle when the puzzle has the same number of characters in all of its rows.
    - Solve the puzzle when the puzzle has more than one character.

4. String print()
    - Print the puzzle when the puzzle is ready.
    - Print the puzzle when the puzzle has the same number of characters in all of its rows.
    - Print the puzzle when the puzzle has more than one character.

Data Flow Cases

1. boolean getDictionary()
    - Generate the dictionary when the puzzle is ready.
    - Generate the dictionary when the puzzle is not ready.
    - Gerate the dictionary when the puzzle is ready.
    - Generate the dictionary where the dictionary has one word and it cannot be found in the boggle puzzle.

- Generate the dictionary where the dictionary has multiple words and no words can be found in the boggle puzzle.
- Generate the dictionary where the dictionary has one word and can be found in the boggle puzzle.
- Generate the dictionary where the dictionary has one word and it cannot be found in the boggle puzzle.
- Generate the dictionary when multiple words are found in the boggle puzzle.

2. boolean getPuzzle()
    - Generate the puzzle when the dictionary is not generated.
    - Generate the puzzle when the dictionary is generated.

3. List<String> solve()
    - Solve the puzzle before generating the puzzle.
    - Solve the puzzle before generating the dictionary.
    - Solve the puzzle once the puzzle is generated but the dictionary is not generated.
    - Solve the puzzle when the dictionary is generated but the puzzle is not generated.

4. String print()
    - Print the puzzle before generating the puzzle.
    - Print the puzzle before generating the dictionary.
    - Print the puzzle once the puzzle is generated but the dictionary is not generated.
    - Print the puzzle when the dictionary is generated but the puzzle is not generated.

External Documentation:

Overview:

The boggle puzzle game is developed to find a set of words defined in a given dictionary from the puzzle.

Files and External Data:
The solution contains five java files and two text files. The two text files are used to fetch the inputs of puzzle and dictionary as BufferedReader streams. The five java files are as follows:

Rushi Samirbhai Patel RS397441@dal.ca
B00886157

1. Main.java :
   This file has the main method which is used to fetch the buffered streams from the text files and call the methods as required.

2. BoggleException.java :
   This is the custom exception class that I created which extends the Exception class to throw the exception in case of any error condition.

3. Boggle.java :
   This is the interface which states the method interfaces of getDictionary(), getPuzzle(), solve() and print().

4. BoggleImplementation.java :
   This class implements the Boggle interface and implements the methods defined in the interface. It is used as the actual implementation of the boggle puzzle game solution.

5. Finder.java :
   This class is used to support the BoggleImplementation class with finder methods. It is used for better modularity of the code.

Data Structures and their relations :

In my solution, I have used Arrays to load my dictionary and puzzle. For the dictionary, I have used String of arrays and to store the puzzle, I have used a 2-D char array. I have also used List as abstract data type and ArrayList as the data structure to return the result of the solved puzzle. The stream as an input to getDictionary method would store the data(words) in the String array and likewise the stream as an input to getPuzzle method would store the data(characters) in a 2-D char array. I have declared them as global static variables so that the values can be used or managed globally.

Key Algorithm and Design patterns:

I have used recursion and backtracking as my key algorithm to solve the boggle puzzle game. As recursion and backtracking causes time issues for a larger set of data, I even optimized my algorithm with the help of a method optimizer() which lets the program know that if the word in the dictionary does not start with a particular character, it will not check all the other possible eight adjacent characters from there on. This managed the time complexity of my code. The solution finds a word using recursion in such a way that it will start with the first character of the puzzle (first character of the first row in the

stream of getPuzzle() method), and will traverse through all the possible characters forming a String of words which will all be checked if they are loaded in the dictionary or not. Once a word is found, I add the word along with its coordinates and the traversal path in the list which is finally returned at the end of the solve() method. Once all the words are found, I sorted the list in ascending order and removed the duplicate words (if found) from the same coordinate or from a different coordinate. As far as the getPuzzle() and getDictioary() methods are concerned, I used the conventional approach to read the stream until the null is encountered and stored the data(words/characters) in a String of array for the dictionary and 2-D char array for the puzzle. To print the puzzle, I looped the 2-D char array puzzle to get the characters and separated them with a tab to better display the puzzle.

Assumptions:

For this solution, I have assumed that:

1. There will not be any whitespaces between two words in a single line in the dictionary stream.
2. If there are duplicate words found in the puzzle having the same X and Y coordinates, the path is considered for either of the words found (in the direction that the word was found) and remove the duplicated word with the path found.
3. I have checked my BufferedReader stream by reading from a file so it is assumed that the stream generated from the file is valid.

References:

[1] "Boggle (Find all possible words in a board of characters) | Set 1 - GeeksforGeeks", 2021. [Online]. Available: https://www.geeksforgeeks.org/boggle-find-possible-words-board-characters/. [Accessed: 11- Nov- 2021].