CSCI 3901 Assignment 4

Fall 2023

Test Cases

Load Board:

Type of the test case	Property of the test case	Property of the input data	Expected result
Input validation	Null check	puzzleStream is null	Return false
	Empty check	puzzleStream is empty	Return false
	Improper Format	puzzleStream is not well formatted	Return false
Boundary case	Just one cell	Board has just one cell	Return false
	Same as word length	Board with same vertical or horizontal cells as the length of the first word	Return true
Control flow	No starting cell	Board with no * (designated starting cell)	Return false
	More than one starting cell	Board with more than one designated start (*)	Return false
	Invalid character	Board with characters apart from the valid words (D, T, ., *, 0-9)	Return false

Dictionary:

Type of the test case	Property of the test case	Property of the input data	Expected result
Input validation	Null check	Passing wordStream as null	Return false
	Empty check	Passing wordStream as empty	Return false

	Improper Format	wordStream is not well formatted	Return false
Boundary case	Min capacity	No words in the dictionary	Return false
	Blank lines	Blank lines between two words	Return false
Control flow	More than one word	A line consisting of more than one space or tab separated words	Return false

Letter Value:

Type of the test case	**Property of the test case**	**Property of the input data**	**Expected result**
Input validation	Null check	Passing valueStream as null	Return false
	Empty Check	Passing valueStream as empty	Return false
	Improper Format	valueStream is not well formatted	Return false
Boundary Case	Min value	Passing no value letter pair	Return false
	Max Value	Passing more than 26 pairs (considering all pairs from a-z are covered)	Return false
Control flow	Tab	The letter value pairs are not one tab separated	Return false
	Letter check	The key (word before tab) is not an alphabet	Return false
	Value check	The value (word after tab) is not numeric	Return false

Print:

Type of the test case	**Property of the test case**	**Property of the input data**	**Expected result**
Data flow	Function call	Calling before loading the board in the system	Return BoardNotLoadedYetException

Place words:

Type of the test case	Property of the test case	Property of the input data	Expected result
Input validation	Empty check	Words arraylist is empty	Return -1
	Null check	Words arraylist is null	Return -1
Boundary case	Min Board space	Board with less horizontal or vertical cells than the length of the first word	Return WordCantFitException
	Words out of boundary	Adding a word which goes out of the boundary of the board	Return -1
Control flow	Words not in dictionary	Trying to place a word which is not in dictionary	Return -1
	Overlapping words	Trying to place a word which overlaps the existing word but still falls withing the boundary and generates max score	Return the respective score generated
	Overwriting words	Trying to place a word which is already placed on the board	Return -1
	Augmented words	After placing a word, if it extends an augmented word which is not in dictionary	Return -1
	Invalid word sequence	Trying to add the list of words that cannot be placed sequentially to form a valid puzzle	Return -1
Data flow	Board not loaded	Calling placeWords before loadBoard method	Return BoardNotLoadedException

	Dictionary not loaded	Calling placeWords before dictionary method	Return DictionaryNotLoadedException
	LetterValue not loaded	Calling placeWords before letterValue method	Return LetterValueNotLoadedException

Word Order:

Type of the test case	**Property of the test case**	**Property of the input data**	**Expected result**
Boundary Check	Min value	Adding only one word to the board	Return a list with one word
Data flow	Board not loaded	Calling placeWords before loadBoard method	Return Empty List
	Dictionary not loaded	Calling placeWords before dictionary method	Return Empty List
	LetterValue not loaded	Calling placeWords before letterValue method	Return Empty List

Steps I have taken to provide some degree of efficiency.

1. I maintained a list of words which are placed on the board (along with their respective metadata including the starting and ending coordinates of the word) which helps me quickly navigate to that word instead of looping through the board again and again.
2. I have maintained a list of augmented words. While checking the placement of a word on the board and verifying if it forms any augmented word, I add those words right at that moment to the list so that I don't have to iterate through the board again to calculate score for augmented words.
3. Calculating the score for augmented words from augmented words lists instead of looping through the board. This can be done by simply summing the letter value of each letter that exists on the list.

External Documentation

Overview

The program is a collection of classes that we've put together to perform a process which includes following step:

1. Have a list of words that are to be added to the puzzle.
2. Load the board in the system. Assign points to letters and add words to the dictionary.
3. Try to add the words to the board which satisfies:
    a. If the word is in the dictionary.
    b. If the word falls within the boundaries of the board.
    c. If the word is not overwriting any existing word on the board.
    d. If after placing a word, it's augmenting a word which is not in the dictionary.
    e. If the placement of the word gives maximum score.

The goal is to build a Scrabble game based upon state space exploration. The objective is to satisfy all puzzle constraints and fit the word to gain maximum points.

Files and external data

All in all, there are 5 classes and 4 exception classes in the project:

1. Main.java:
    a. Contains the main method and basically holds the driver code.
    b. Handles the running of each method in the system.
2. NewWord.java
    a. Holds the structure of any word which is placed on the board.
    b. Stores the metadata of the word such as:
        i. Start Index
        ii. End Index
        iii. Score
    c. This metadata helps in quickly navigating to the word on the board and makes program efficient.
3. PlaceWord.java:
    a. Holds the main logic for placing a word on the board.
    b. Handles multiple operations including:
        i. Checking if the word is in the dictionary.
        ii. Checking if the word falls withing the boundaries of the board.
        iii. Checking if the word is not overwriting any existing words on the board.
        iv. Checking if after placing a word, will it extend any augmented word.
        v. Checking the best placement of the word to generate the maximum points.
4. ValidateStream.java
    a. All the validation logic for all the streams used within the program.
    b. Validates the board, dictionary, and letter value stream.
5. WordPlacement.java
    a. Consists of the implementation of all methods used within the project.
6. BoardNotLoadedException.java: This exception is triggered when the board is not loaded in the system yet.
7. DictionaryNotLoadedException.java: This exception is triggered when the dictionary is not

loaded in the system yet.

8. LetterValueNotLoadedException.java: This exception is triggered when the letter value pair are not loaded in the system yet.
9. WordCantFitException.java: This exception is triggered when the first word does not fit on the board vertically nor horizontally.

Data structures and their relations to each other

Mostly, the entire project depends upon one 2-D array (board).
char[][] board:

1. The **board** is a 2D character array that serves as the central data structure for representing the Scrabble game board.
2. It is structured as a grid, with each cell in the grid containing either a letter, a digit, or a special character, which collectively forms the puzzle layout.
3. The purpose of the **board** is to provide a visual and structural representation of the game, allowing words to be placed on it.
4. It plays a critical role in word placement and scoring. As words are placed on the board, the letters interact with existing tiles, word multipliers, and letter multipliers to calculate the score for each word.
5. As The **board** is updated dynamically as words are placed, and the overall puzzle layout evolves during the game.
6. It forms the canvas on which the Scrabble game is played, and the state of the **board** determines the score and the feasibility of word placements.

Set<String> dictionarySet:

Represents the dictionary of valid words that can be placed on the board.
Maintained as a set to quickly check if a word is valid and part of the allowed vocabulary.

Map<Character, Integer> letterValueMap:

1. Stores letter values for scoring, where each character corresponds to a letter, and the associated integer is its scoring value.
2. Used to calculate the score for each placed word.

Assumptions

1. loadBoard: The number of columns/number of characters in the first row should be consistent throughout the other rows/lines.
2. dictionary: There won't be any black lines between the words.
3. letterValue: There won't be any black lines between two letter value pairs.

Choices

placeWords: If the word has similar score for both vertical and horizontal position, by default it'll be placed in horizontal position and that'd be consistent throughout.

Key algorithms and design elements

Place Word:

1. It searches for the first word placement (positioned at '*') and checks for horizontal and vertical placements.
2. It considers the one with the highest score and places the word on the board.
3. The greedy approach aims to maximize the score for each word placement. It evaluates different placement options and selects the one with the highest score. It considers both horizontal and vertical placements, prioritizing the one with the highest score.
4. If there's at least one word on the board, it searches for possible placements based on common letters. It calculates and compares the scores of horizontal and vertical placements, choosing the one with the higher score. It updates the board with the placed word and keeps track of the placement's metadata.
5. If no direct word placement is possible, the code checks for cross-placements (words placed perpendicular to existing words) by checking if common letters exist. It looks for valid cross-placements and calculates the scores for these words.
6. Along with all of that, before placing a word, it checks for all constrains including:
    a. Dictionary word validation
    b. Augmented score
    c. Boundary check
    d. Overwrite check.

Limitations

1. The code is designed for single-player use, and it doesn't support multiplayer features, such as scoring for multiple players or handling simultaneous moves from different participants.
2. In complex and lengthy games, its performance may degrade, and it may not be suitable for real-time or multiplayer applications with multiple simultaneous moves.