Sfwr Eng 2XB3 - The Scrabbler

Group 33
Greg Barkans
Curtis Milo
Colin Gillespie

Prof: Dr. Samvi
Lab: L01

Contents

1 MIS (Public API)

1.1 Class BST

public BST()	Creates an empty Binary search tree
public void insert(int key, Word value)	Given a key for the length of the string, the string value will be associated with the key given; values with the same key will not be over written.
public Word search(int key, String word)	Given a key of the length of the string and the word we wish to find, we will return the reference to an object that holds both the word itself and the score that is associated with the word.
public boolean isWord(int key, String word)	Determines if the string inputted belongs to our list of words.
public Word[] matchesPattern(int key, String[] regExpArray)	Given the length of the string in key, will return all the strings where each string character matches the same index of regExpArray.

1.2 Class TrieST

public TrieST()	Creates an empty trie with the first node being empty
public void insertWord(Word word)	Will add the given word in the proper location into the trie.
public Word search(String word)	Will return the Word abstract data type that is associated with the string word
public Word[] matchPattern(String[] regExpArray)	Returns all the words in the trie that match the pattern of the regular expression

1.3 Class Word

public Word(String word)	Creates an object with 2 fields, one that stores the String word and one that is the score of the object based on the rules of the scrabble board.
public int setScore(int val)	Sets the score value of the word to the desired value.

1.4 Class Model

public Model()	Creates a new Model object, storing information of the board like what tiles are currently on the grid, what spaces we have highlighted, and holds the list of words that are valid.
public Word[] matchTiles()	Creates an array of the Words, This will represent the words that we can make given the state of the board and the users tiles.
public String[] createRegex()	Will create an array where each element will correspond to a regular expression for one character of a word. This is determined based on what pieces are currently on the board.
public Word[] getMatches(int number)	Creates an array of size number or less in a sorted order where the 0 index will have the highest score value.

1.5 Class Controller

public void Controller()	Creates a new controller for the interface.html page, this is only instantiated when the page loads.
public static void onGridClick(htmlClickEvent e)	Depending on the state the controller this will either add either ask to input a letter or will highlight the square that was clicked.
public static void onNextClick()	Will change the state of the website one forward and will update the view to match our new state.
public static void onBackClick()	Will change the state of the website one back and will update the view to match our new state.
public void setTiles()	Will retrieve the inputted tiles from the HTML input boxes and will set the field of in model to be a list of these tiles.
public void resetN()	Button event handler to handle when we click the reset button. This will simply refresh the page.

1.6 Class View

public View(Canvas canvasID)	Uses the canvas given to create a new 7x7 scrabble board view.
public void addLetter(char letter, int x, int y)	Will add a Letter to our canvas at the position of x and y.
public void addHLetter(char letter,int x, int y)	Adds a highlighted to the screen.
public void highlight(char letter, int x, int y)	Adds a black highlighted square to the screen.
public void updatePStatus(String string)	Updates the message displayed to the user with the string passed in.
public void updateResult(String[] results)	Updates the result message box with the values from results.

2 MID

2.1 Class BST

A BST That is used to separate the strings of different lengths. The value of the BST is a trie symbol table that will hold all of the words of this length

Fields	Node root: The starting Node of the BST
private class Node	• The length of the strings affiliated with the trie. • Two Node references for key values less than and key values greater then key. • The value of this node will be a Trie which holds all the words of the given length.
private void insertREC(Node node, int key, Word value)	Will recursively search for the Node associated with with the inputted key. Once they key is found it will check to see if we have an undefined trie symbol table then we will instantiate one. Then we call the insert method for the trie symbol table to add the word.
private Word[] matchPatternREC(Node node, int key, String[] regExpArray)	Will recursively find the node associated with our key, Then will call the trie symbol table matchPattern method to return any word such that all of the characters match each element of the regular expression array.
private TrieST searchREC(Node node, int key)	Recursively find the Node associated with the key and will then return the trie symbol table associated with Node.

2.2 Class TrieST

An Implementation of a trie symbol table where the root of table will have a null character value. The table has an an array of children where each node will have a node associated with a letter [a-z]. If this is a valid word in our dictionary then we will have Word Object within wordVal.

Fields	RNode root: The starting RNode of the trie symbol table, where the character value is the null character.
private class RNode	• The character at the position the height from the root. • An array of the references to Nodes that hold character next in our String. • If the sequence to get to this node and this nodes value create a word, then the field wordVal will Store a Word Object.
private void insertWordREC(RNode node, Word wordInput,int charAT)	Will first properly define the current Node if we have any null references for: The Node itself, The Node's children and there values. Then if we are at the length of our string, then we will set the value of wordVal to the input Word. If not then we will find the next Node to visit and preform recursion on this Node.
private Word searchREC(RNode node, String wordInput,int charAT)	Recursively will find the Node with a path associated with wordInput. Then we will return the value of wordVal (returns null if it is not a word).
private void matchPatternREC(RNode node, Word[] combinations, String[] regExpArray, int charAT)	Checks each child to see if the regular expression regExpArray[charAT] matches the values any of the children of a this Node. If the pattern then we will preform recursion on this Node to see if the next regular expressions matches. Our base case is either do not have a value or any more children where we simply return, or when we have a reached the end of our regular expression array, if we do not have a Word at this node we will return but if we do have a Word then will add this value to our array.

2.3 Class Word

Fields	• String word: The string of the word that we are storing. • int score: The number value that we assign to the word.
private class scoreRules	• Regular Expression regEX: A regular expression for the letters that fall into this word. • int score: The score pertaining to this regular expression.
private int getScore(String word)	Creates score rules for every letter in the alphabet for the rules scrabble. Then will some the total score for the string where for every characters in word such that there exists a rule for that letter.

2.4 Class Model

Fields	• String [ROWS][COLUMNS] grid: holds the character held on every space of the board. • String [ROWS][COLUMS] hl_grid: hold if the space of the grid is highlighted or not. • Word[] words: holds all of the words that match our current state of the board once the onNextClick has reached the result state. • Heap heap: used to implement a priority queue for the our list of words. • char[] tiles: holds all of the tiles the user has once the onNextClick has reached the result state.
public Object createRegex()	(TODO: Greg explains how it works)
public void sortMatches(Word[] matches)	Adds all of the Words that match the conditions to our heap.
private String grabWord(int i, int j, String dir)	Will grab the array of characters that are perpendicular to the highlighted tiles
private RegExp modifyRegex(int i, int j, String d, String reg)	(TODO: Greg explain how it works, consider renaming variables)
private boolean adj(int i, int j, String d)	Checks for an adjacent highlight in the same direction.
private Object findHl()	Returns the fist occurrence of a highlighted piece and if the direction is vertical, horizontal or single.
private void initGrid(int rows,int columns)	Creates a grid of size rows X columns initialized to to an empty square.
private BST getDictionary()	Reads in from the data sheet of possible words for each word, it will create a new Word Object and will insert it into the BST.

2.5 private class Heap

Uses this.word from parent model

private Heap()	Set's our parent field this.word to an empty heap.
private int size()	Returns the number of elements within the heap.
private void exch(int i, int j)	Swaps the Words at positions i and j.
private void less(int i, int j)	Returns if the score of our score at position i is less then our word at position j.
private void add(Word wordADT)	Adds a new Item to the bottom of the heap, then swims the Word to the proper location in the heap
private Word top()	Will return the our largest element on our heap if our heap is not empty.
private Word pop()	Will return and delete the our largest element on our heap if our heap is not empty. We first move the max node to the bottom, remove the max from the heap, then sink our new top Word.
private void sink(int index)	Will compare the parent Word at index to its children and will swap and repeat if the score of one of the children is larger then the parent
private void swim(int index)	Will compare the child Word at index to its parent and will swap and repeat if the score of one of the parent is smaller then the child

2.6 Class Controller

Fields	• Canvas canvasID: the reference to the Canvas on the html, used to draw the grid • Button nextButton: the reference to the Next Button on the HTML. • Button backButton: the reference to the Back Button on the HTML. • Button resetButton: the reference to the Reset Button on the HTML.
public void onGrid-Click(htmlClickEvent e)	• Letters State: Asks for input for the tile that was clicked on • Highlight State: Will highlight or remove the highlighting on the grid square clicked on
public void onNextClick()	• Letters State: Change the State to Highlight and update the View • Highlight State: Will move to the Tiles state if there is a valid highlighted row on the board • Tiles: Checks to see if all the tiles are valid, if they are then we will move to the Result state. • Result: will retrieve all of the valid words for this location and will update the View to display them

public void onBackClick()	• Letters State: Nothing • Highlight State: Change state to Letters • Tiles: Change state to Highlight • Result: Change state to Tiles
public boolean tileCheck()	Returns true if all of the inputs for the tiles are one character that is a letter or - or ?
private boolean hl_check()	Returns true if we have between 1 and 7 spaces highlighted in a row.
private String getLetter(int x_Pos, int y_Pos)	Returns The String that is in the location of x_Pos, Y_Pos on the grid.
private Object getClick(htmlClickEvent e)	gets the position on the grid that was clicked by using the normalize function. Returns a pair of x and y.
private String getLetter(int x_Pos, int y_Pos)	Returns the modified numbers of where the user clicked such that it corresponds to the coordinate system of the grid arrays. Removing such things like offsets and grid size to get a how number between [0,6].