# Wordle Report

## UML Diagram

Diagram

Description automatically generated

## Code

#### CLIWordle.java

import java.io.IOException;  
import java.util.Scanner;  
  
public class CLIWordle {  
  
 public static void main(String[] args) throws IOException {  
 Scanner scanner = new Scanner(System.in);  
 int randomMode, spoilerMode, strictMode;  
  
 // Choose various game modes to play  
 do {  
 System.out.println("Press 1 for a random word or 2 for a fixed word");  
 randomMode = scanner.nextInt();  
 } while (randomMode != 1 && randomMode != 2);  
 do {  
 System.out.println("Press 1 for spoiler mode or 2 for secret mode");  
 spoilerMode = scanner.nextInt();  
 } while (spoilerMode != 1 && spoilerMode != 2);  
 do {  
 System.out.println("Press 1 for strict mode or 2 for loose mode");  
 strictMode = scanner.nextInt();  
 } while (strictMode != 1 && strictMode != 2);  
 Model model = new Model(randomMode, spoilerMode, strictMode);  
  
 gameLoop(model, scanner);  
 }  
  
 public static void gameLoop(Model model, Scanner scanner) {  
 String guess;  
 int i;  
 if (model.isSpoilerFlag()) System.out.println("The answer is " + model.letterListToString(model.getAnswer()));  
 while (model.getGameFlag()) {  
 // Enter guess  
 System.out.println("Enter guess number " + (model.getTurnCount()+1));  
 guess = scanner.next();  
 while (!model.isValidWord(guess)) {  
 System.out.println("Not a valid guess. Try again!");  
 guess = scanner.next();  
 }  
  
 model.submitWord(guess);  
 String indicators = model.getIndicators();  
 model.submitGuess();  
 System.out.println("");  
  
 // Print out results  
 if (model.getWinFlag()) {  
 System.out.println("Correct! You won!");  
 return;  
 }  
 else {  
 System.out.println(indicators);  
 System.out.println(guess);  
 System.out.println("");  
 System.out.println("Correct letters: " + model.letterListToStringList(model.getCorrectLetters()));  
 System.out.println("Partial letters: " + model.letterListToStringList(model.getPartialLetters()));  
 System.out.println("Wrong letters: " + model.letterListToStringList(model.getWrongLetters()));  
 System.out.println("Unused letters: " + model.letterListToStringList(model.getUnusedLetters()));  
 System.out.println("");  
 }  
 }  
 System.out.println("No more guesses allowed. Better luck next time!");  
 }  
}

#### GUIWordle.java

import java.io.IOException;  
import java.util.Scanner;  
  
public class GUIWordle {  
  
 public static void main(String[] args) throws IOException {  
 Scanner scanner = new Scanner(System.in);  
 int randomMode, spoilerMode, strictMode;  
  
 // Choose various game modes to play  
 do {  
 System.out.println("Press 1 for a random word or 2 for a fixed word");  
 randomMode = scanner.nextInt();  
 } while (randomMode != 1 && randomMode != 2);  
 do {  
 System.out.println("Press 1 for spoiler mode or 2 for secret mode");  
 spoilerMode = scanner.nextInt();  
 } while (spoilerMode != 1 && spoilerMode != 2);  
 do {  
 System.out.println("Press 1 for strict mode or 2 for loose mode");  
 strictMode = scanner.nextInt();  
 } while (strictMode != 1 && strictMode != 2);  
  
 Model model = new Model(randomMode, spoilerMode, strictMode);  
 if (model.isSpoilerFlag()) System.out.println("The answer is \"" + model.letterListToString(model.getAnswer()) + "\"");  
 Controller controller = new Controller(model);  
 View view = new View(model, controller);  
 }  
}

#### View.java

import javax.swing.\*;  
import java.awt.\*;  
import java.awt.event.ActionEvent;  
import java.util.\*;  
import java.util.List;  
  
public class View implements Observer {  
 private final Model model;  
 private final Controller controller;  
 private JFrame frame;  
 private List<JButton> keyboard;  
 private List<List<JLabel>> guesses;  
 private JLabel errorMessage;  
 private JLabel answerMessage;  
 private JButton newGameBtn;  
 private int head;  
  
  
 public View(Model model, Controller controller) {  
 this.model = model;  
 this.controller = controller;  
 List<Color> colours = controller.getColours();  
 model.addObserver(this);  
 this.frame = initialise();  
 this.controller.initialise(this);  
 }  
  
 public JFrame initialise() { // General function to initialise the view for a new game  
 this.head = 0;  
 JFrame frame = new JFrame("Wordle Game");  
 try{  
 UIManager.setLookAndFeel(UIManager.getCrossPlatformLookAndFeelClassName());  
 }catch(Exception e){  
 e.printStackTrace();  
 }  
 frame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);  
 initLabels(frame);  
 initButtons(frame);  
 frame.setSize(500, controller.getNumGuesses()\*50 + 250);  
 frame.setLayout(null); //using no layout managers  
 frame.setVisible(true);  
 return frame;  
 }  
  
 private void initLabels(JFrame frame) { // Loads in the labels used to show user input letters  
 this.guesses = new ArrayList<>();  
 int y = 25;  
 for (int i = 0; i < controller.getNumGuesses(); i++) {  
 List<JLabel> row = new ArrayList<>();  
 this.guesses.add(row);  
 int x = 70;  
 for (int j = 0; j < 5; j++) {  
 JLabel label = new JLabel();  
 label.setBounds(x,y,50,50);  
 label.setOpaque(true);  
 label.setHorizontalAlignment(SwingConstants.CENTER);  
 label.setVerticalAlignment(SwingConstants.CENTER);  
 label.setBackground(Color.lightGray);  
 frame.add(label);  
 this.guesses.get(i).add(label);  
 x += 60;  
 }  
 y += 60;  
 }  
  
 this.errorMessage = new JLabel("<html>"+ "Invalid Word!" +"</html>");  
 this.answerMessage = new JLabel("<html>"+ "The answer was " + controller.getAnswerAsString() +"</html>");  
 this.errorMessage.setForeground(Color.red);  
 this.answerMessage.setForeground(Color.red);  
 this.errorMessage.setBounds(420, 10, 70, 200);  
 this.answerMessage.setBounds(420, 100, 70, 200);  
 this.errorMessage.setVisible(false);  
 this.answerMessage.setVisible(false);  
 frame.add(this.errorMessage);  
 frame.add(this.answerMessage);  
 }  
  
 private void initButtons(JFrame frame) { // Loads all the buttons  
 // Load in the keyboard  
 this.keyboard = new ArrayList<>();  
 int keyboardHeightConst = controller.getNumGuesses()\*50 + 100;  
 int x = 20;  
 int y = keyboardHeightConst;  
 String[] qwerty = "qwertyuiopasdfghjklzxcvbnm".split("");  
 for (int i = 0; i < qwerty.length; i++) {  
 JButton btn = new JButton(qwerty[i]);  
 btn.setBounds(x,y,30,30);  
 btn.setMargin(new Insets(0, 0, 0, 0));  
 btn.addActionListener((ActionEvent e) -> {keyboardBtnHandler(btn.getText());});  
 btn.setFont(new Font("Arial", Font.PLAIN, 10));  
 btn.setBackground(Color.lightGray);  
 this.keyboard.add(btn);  
 frame.add(this.keyboard.get(i));  
 if (qwerty[i].equals("p") || qwerty[i].equals("l")) {  
 if (qwerty[i].equals("p")) x = 35;  
 else x = 60;  
 y += 40;  
 } else x += 40;  
 }  
 // Load in enter and delete buttons  
 JButton enter = new JButton("ENT");  
 JButton del = new JButton("DEL");  
 enter.setBounds(425, keyboardHeightConst+ 65, 60, 40);  
 del.setBounds(425, keyboardHeightConst+ 15, 60, 40);  
 enter.setFont(new Font("Arial", Font.BOLD, 10));  
 del.setFont(new Font("Arial", Font.BOLD, 10));  
 enter.addActionListener((ActionEvent e) -> {enterHandler();});  
 del.addActionListener((ActionEvent e) -> {delHandler();});  
 frame.add(enter);  
 frame.add(del);  
  
 // Load in new game button  
 this.newGameBtn = new JButton("New Game");  
 this.newGameBtn.setBounds(385, keyboardHeightConst-70, 100, 40);  
 this.newGameBtn.addActionListener((ActionEvent e) -> {newGameHandler();});  
 this.newGameBtn.setVisible(false);  
 frame.add(this.newGameBtn);  
 }  
  
 public void addLetter(Letter letter) { // Letter pressed by user is shown in corresponding label  
 JLabel label = this.guesses.get(controller.getTurnCount()).get(head);  
 label.setText(letter.getName());  
 if (head < 5) head++;  
 this.frame.repaint();  
 }  
  
 public void removeLetter() { // Make last updated label blank  
 if (head > 0) head--;  
 JLabel label = this.guesses.get(controller.getTurnCount()).get(head);  
 label.setText(null);  
 this.frame.repaint();  
 }  
  
 private void keyboardBtnHandler(String text) { // Event handler for when the letters of the keyboard are pressed  
 if (controller.getGameFlag()) controller.submitLetter(text);  
 }  
  
 private void enterHandler() { // Event handler for when the enter key is pressed  
 if (controller.getGameFlag()) {  
 this.errorMessage.setVisible(!controller.isValidWord(controller.getCurrentGuessAsString())  
 && controller.getStrictFlag());  
 controller.submitGuess();  
 }  
 }  
  
 private void delHandler() { // Enter handle for when the delete key is pressed  
 if (controller.getGameFlag()) controller.removeLetter();  
 }  
  
 private void newGameHandler() {  
 controller.newGame();  
  
 for (List<JLabel> row : this.guesses)  
 for (JLabel label : row) {  
 label.setText(null);  
 label.setBackground(Color.lightGray);  
 label.setForeground(Color.black);  
 }  
  
 for (JButton btn : this.keyboard) {  
 btn.setBackground(Color.lightGray);  
 btn.setForeground(Color.black);  
 }  
  
 this.newGameBtn.setVisible(false);  
 this.answerMessage.setVisible(false);  
 }  
  
 @Override  
 public void update(Observable o, Object arg) {  
 // Update background of labels that have been used  
 List<List<Letter>> submittedGuesses = controller.getSubmittedGuesses();  
 List<Color> colours = controller.getColours();  
 this.head = 0;  
 List<Letter> guess = submittedGuesses.get(controller.getTurnCount()-1);  
 for (int i = 0; i < guess.size(); i++) {  
 JLabel label = this.guesses.get(controller.getTurnCount() - 1).get(i);  
 Color colour = colours.get(guess.get(i).getRoundStates()[i]);  
 label.setBackground(colour);  
 if (guess.get(i).getRoundStates()[i] == Model.WRONG) label.setForeground(Color.white);  
 }  
  
 // Update backgrounds of buttons  
 for (JButton btn : this.keyboard) {  
 Letter letter = controller.getLetter(btn.getText());  
 if (letter.getButtonState() != Model.UNASSIGNED) {  
 Color colour = colours.get(letter.getButtonState());  
 btn.setBackground(colour);  
 if (letter.getButtonState() == Model.WRONG) btn.setForeground(Color.white);  
 }  
 }  
  
 // Reveal new game button after first turn  
 if (model.getTurnCount() == 1) this.newGameBtn.setVisible(true);  
 // Reveal word if player runs out of turns  
 if (!controller.getGameFlag() && !controller.getWinFlag()) answerMessage.setVisible(true);  
 }  
  
  
}

#### Controller.java

import java.awt.\*;  
import java.util.ArrayList;  
import java.util.List;  
  
public class Controller {  
 private final Model model;  
 private View view;  
  
 public Controller(Model model) {  
 this.model = model;  
 }  
  
 public void initialise(View view) {  
 this.view = view;  
 }  
  
 public int getNumGuesses() {  
 return Model.NUMGUESSES;  
 }  
  
 public void newGame() {  
 model.initialise();  
 if (model.isSpoilerFlag()) System.out.println("The answer is \"" + model.letterListToString(model.getAnswer()) + "\"");  
 }  
  
 public List<Color> getColours() { // Get list of colours with indicies matching letter state values  
 List<Color> colours = new ArrayList<>();  
 colours.add(Color.lightGray); // Model.UNASSIGNED  
 colours.add(Color.darkGray); // Model.WRONG  
 colours.add(Color.yellow); // Model.PARTIAL  
 colours.add(Color.green); // Model.CORRECT  
 return colours;  
 }  
  
 public void submitLetter(String text) { // Submit letter to model and call view to update label  
 if (model.getCurrentGuess().size() < 5) {  
 Letter letter = model.getLetter(text);  
 model.submitLetter(letter);  
 view.addLetter(letter);  
 }  
 }  
  
 public void removeLetter() { // Remove letter in model and clear corresponding label  
 if (model.getCurrentGuess().size() > 0) {  
 model.removeLastLetter();  
 view.removeLetter();  
 }  
 }  
  
 public String getCurrentGuessAsString() {  
 return model.letterListToString(model.getCurrentGuess());  
 }  
  
 public String getAnswerAsString() {  
 return model.letterListToString(model.getAnswer());  
 }  
  
 public boolean isValidWord(String word) {  
 return model.isValidWord(word);  
 }  
  
 public void submitGuess() {  
 if (model.getCurrentGuess().size() == 5 && model.isCurrentGuessValid()) {  
 model.submitGuess();  
 }  
 }  
  
 public List<List<Letter>> getSubmittedGuesses() {  
 return model.getSubmittedGuesses();  
 }  
  
 public int getTurnCount() {  
 return model.getTurnCount();  
 }  
  
 public boolean getWinFlag() {  
 return model.getWinFlag();  
 }  
  
 public boolean getGameFlag() {  
 return model.getGameFlag();  
 }  
  
 public boolean getStrictFlag() { return model.getStrictFlag(); }  
  
 public Letter getLetter(String l) {  
 return model.getLetter(l);  
 }  
}

#### Model.java

import java.io.\*;  
import java.util.\*;  
  
public class Model extends Observable {  
 // Constants  
 private static final String ANSWER\_FILE = "src/assets/common.txt";  
 private static final String WORDS\_FILE = "src/assets/words.txt";  
 public static final int UNASSIGNED = 0;  
 public static final int WRONG = 1;  
 public static final int PARTIAL = 2;  
 public static final int CORRECT = 3;  
 public static final int NUMGUESSES = 6;  
 public static String FIXED\_ANSWER = "undid";  
 // Flags  
 private final boolean spoilerFlag; // Sets whether the game will reveal the answer to the user  
 private final boolean randomFlag; // Sets whether the game generates an answer or is provided an answer by user  
 private final boolean strictFlag; // Sets whether the user can input any five-letter words or not  
 private boolean gameFlag; // When true, game is still in play  
 private boolean winFlag; // When true, player has won the game  
 // Game attributes  
 private final List<String> answers; // List of valid answers  
 private final List<String> words; // List of valid guesses  
 private HashMap<String, Letter> alphabet; // List of Letter objects in the game  
 private int turnCount; // Which turn the game is on  
 private List<Letter> answer; // The answer the user needs to guess to win  
 private List<Letter> currentGuess; // Most recent guess submitted by user  
 private List<List<Letter>> submittedGuesses; // List of all submitted guesses  
 public List<Letter> unusedLetters; // List of unused letters  
 public List<Letter> wrongLetters; // List of incorrectly guessed letters  
 public List<Letter> partialLetters; // List of letters that are in the answer but wrong position  
 public List<Letter> correctLetters; // List of correctly guessed letters  
  
 public Model(int randomMode, int spoilerMode, int strictMode) throws IOException {  
 // Initialise answer list  
 BufferedReader ar = new BufferedReader(new FileReader(ANSWER\_FILE));  
 this.answers = new ArrayList<>();  
 String line;  
 while((line = ar.readLine()) != null) { this.answers.add(line); }  
 ar.close();  
 // Initialise words list  
 BufferedReader wr = new BufferedReader(new FileReader(WORDS\_FILE));  
 this.words = new ArrayList<>();  
 while((line = wr.readLine()) != null) { this.words.add(line); }  
 this.words.addAll(this.answers);  
 wr.close();  
 // Set attributes  
 this.randomFlag = randomMode == 1;  
 this.spoilerFlag = spoilerMode == 1;  
 this.strictFlag = strictMode == 1;  
 initialise();  
 }  
  
 /\*\*  
 \* Initialises the game attributes  
 \* @pre. Words and Answers have been successfully initialised  
 \* @post. this.answer is a five-letter Letter list  
 \*/  
 public void initialise() {  
 assert !this.words.isEmpty() && !this.answers.isEmpty();  
 this.winFlag = false;  
 this.gameFlag = true;  
 if (this.turnCount > 0) resetAlphabet();  
 else this.alphabet = createAlphabet();  
 this.unusedLetters = new ArrayList<>(this.alphabet.values());  
 this.submittedGuesses = new ArrayList<>();  
 this.wrongLetters = new ArrayList<>();  
 this.partialLetters = new ArrayList<>();  
 this.correctLetters = new ArrayList<>();  
 this.currentGuess = new ArrayList<>();  
 this.turnCount = 0;  
 this.answer = generateAnswer();  
 int i = 0;  
 for (Letter l : this.answer) { // Update the letters in alphabet  
 assert l != null;  
 l.addPos(i);  
 l.setInGame();  
 i++;  
 }  
 assert this.answer.size() == 5;  
 }  
  
  
 private HashMap<String, Letter> createAlphabet() {  
 String[] letters = "abcdefghijklmnopqrstuvwxyz".split("");  
 HashMap<String, Letter> alphabet = new HashMap<>();  
 for (String c : letters) alphabet.put(c, new Letter(c));  
 return alphabet;  
 }  
  
 private void resetAlphabet() {  
 String[] letters = "abcdefghijklmnopqrstuvwxyz".split("");  
 for (String c : letters) {  
 Letter l = getLetter(c);  
 l.initialise();  
 }  
 }  
  
 private List<Letter> generateAnswer() {  
 String a;  
 List<Letter> answer = new ArrayList<>();  
  
 // If random mode is on, generate the answer by picking a random word from answer list  
 if (this.randomFlag) {  
 Random rand = new Random();  
 a = this.answers.get(rand.nextInt(this.answers.size()));  
 String[] sa = a.split("");  
 for (String s : sa) answer.add(this.alphabet.get(s));  
 }  
 // Else, set the fixed word, check it's valid, and then set that as the answer  
 else {  
 a = FIXED\_ANSWER;  
 String[] sa = a.split("");  
 for (String s : sa) answer.add(this.alphabet.get(s));  
 }  
 assert answer.size() == 5; // Answer was initialised with a 5-letter word  
 return answer;  
 }  
  
 /\*\*  
 \* Get letter from alphabet  
 \* @pre. l is a single alphabetic letter  
 \* @post. None  
 \*/  
 public Letter getLetter(String l) {  
 l = l.toLowerCase();  
 assert l.length() == 1; // Key is length of 1  
 assert l.matches("[a-z]"); // Key is an alphabetic value  
 return this.alphabet.get(l);  
 }  
  
 /\*\*  
 \* Returns a boolean corresponding to if currentGuess is valid  
 \* @pre. None  
 \* @post. None  
 \*/  
 public boolean isCurrentGuessValid() {  
 return isValidWord(String.join("", letterListToStringList(this.currentGuess)));  
 }  
  
 /\*\*  
 \* Checks input guess to see if it is valid  
 \* @pre. this.words contains all words from WORDS\_FILE, this.strictFlag is set  
 \* @post. None  
 \*/  
 public boolean isValidWord(String guess) {  
 if (!guess.matches("[a-z]+")) {  
 System.out.println("Word is not alphabetic!");  
 return false;  
 }  
 if (guess.length() != 5) {  
 System.out.println("Word is not five characters!");  
 return false;  
 }  
 if (this.strictFlag) if (!this.words.contains(guess)) { // If strictFlag is false, no need to check guess list  
 System.out.println("Word is not in the guess list");  
 return false;  
 }  
 return true;  
 }  
  
 private boolean isCorrectGuess() {  
 for (int i = 0; i < 5; i++) {  
 if (this.currentGuess.get(i) != this.answer.get(i))  
 return false;  
 }  
 return true;  
 }  
  
 /\*\*  
 \* Appends an input Letter to currentGuess  
 \* @pre. letter is in the alphabet, this.currentGuess is less than size 5  
 \* @post. this.currentGuess contains at least one instance of letter  
 \*/  
 public void submitLetter(Letter letter) {  
 assert alphabet.containsValue(letter);  
 assert this.currentGuess.size() < 5;  
 this.currentGuess.add(letter);  
 assert this.currentGuess.contains(letter);  
 }  
  
 /\*\*  
 \* Removes the last Letter from currentGuess  
 \* @pre. this.currentGuess is not size 0  
 \* @post. this.currentGuess won't be a complete word  
 \*/  
 public void removeLastLetter() {  
 assert this.currentGuess.size() > 0;  
 this.currentGuess.remove(this.currentGuess.size() - 1);  
 assert this.currentGuess.size() < 5;  
 }  
  
 /\*\*  
 \* Submits a five-letter word guess one letter at a time  
 \* @pre. word is a five-letter string  
 \* @post. this.currentGuess is a list containing five Letter elements  
 \*/  
 public void submitWord(String word) {  
 assert word.length() == 5;  
 clearGuess();  
 String[] w = word.split("");  
 for (String l : w) submitLetter(this.alphabet.get(l));  
 assert this.currentGuess.size() == 5;  
 }  
  
 /\*\*  
 \* Submits currentGuess as final guess for the turn  
 \* @pre. this.currentGuess is a valid five-letter guess,  
 \* @post. this.submittedGuesses has at least one element  
 \*/  
 public void submitGuess() {  
 assert isCurrentGuessValid();  
 this.submittedGuesses.add(this.currentGuess);  
 updateAlphabet();  
 this.turnCount++;  
 this.winFlag = isCorrectGuess();  
 if (this.turnCount >= NUMGUESSES) this.gameFlag = false;  
 else if (this.winFlag) this.gameFlag = false;  
 setChanged();  
 notifyObservers();  
 clearGuess();  
 assert this.submittedGuesses.size() > 0;  
 }  
  
 private void updateAlphabet() {  
 List<Letter> word = this.currentGuess;  
 int i = 0;  
 for (Letter l : word) {  
 this.unusedLetters.remove(l);  
 setButtonState(l,i);  
 switch (l.getButtonState()) {  
 case WRONG:  
 if (!this.wrongLetters.contains(l)) this.wrongLetters.add(l);  
 break;  
 case PARTIAL:  
 if (!this.partialLetters.contains(l)) this.partialLetters.add(l);  
 break;  
 case CORRECT:  
 if (!this.correctLetters.contains(l)) {  
 this.partialLetters.remove(l);  
 this.correctLetters.add(l);  
 } break;  
 default:  
 this.unusedLetters.add(l);  
 break;  
 }  
 l.setHasBeenUsed(true);  
 i++;  
 }  
 }  
  
 private void setButtonState(Letter letter, int i) {  
 if (!letter.hasBeenUsed()) {  
 if (letter.isInGame()) {  
 if (letter.getPosList().contains(i)) letter.setButtonState(CORRECT);  
 else letter.setButtonState(PARTIAL);  
 }  
 else letter.setButtonState(WRONG);  
 }  
 else if (letter.getButtonState() == PARTIAL) {  
 if (letter.getPosList().contains(i)) letter.setButtonState(CORRECT);  
 }  
 }  
  
 /\*\*  
 \* Create a string indicating whether each guess letter is correct, partial, or wrong  
 \* @pre. this.currentGuess is a valid five-letter word  
 \* @post. indicators is the same length as this.currentGuess  
 \*/  
 public String getIndicators() {  
 assert isCurrentGuessValid();  
 StringBuilder indicators = new StringBuilder();  
 for (int i = 0; i < this.currentGuess.size(); i++) {  
 Letter l = this.currentGuess.get(i);  
 if (l.getRoundStates()[i] == CORRECT) indicators.append("o");  
 else if (l.getRoundStates()[i] == PARTIAL) indicators.append("~");  
 else if (l.getRoundStates()[i] == WRONG) indicators.append("x");  
 }  
 assert indicators.length() == this.currentGuess.size();  
 return indicators.toString();  
 }  
  
 /\*\*  
 \* Converts a List<Letter> to a List<String>  
 \* @pre. None  
 \* @post. None  
 \*/  
 public List<String> letterListToStringList(List<Letter> letters) {  
 List<String> stringList = new ArrayList<>();  
 for (Letter l : letters) stringList.add(l.getName());  
 return stringList;  
 }  
  
 /\*\*  
 \* Converts a List<Letter> to a String  
 \* @pre. None  
 \* @post. None  
 \*/  
 public String letterListToString(List<Letter> letters) {  
 return String.join("",letterListToStringList(letters));  
 }  
  
 /\*\*  
 \* Converts a String to a List<Letter>  
 \* @pre. words contains only characters in the alphabet  
 \* @post. None  
 \*/  
 public List<Letter> stringToLetterList(String word) {  
 assert word.matches("[a-zA-Z]+");  
 word = word.toLowerCase();  
 List<Letter> letters = new ArrayList<>();  
 for (char c : word.toCharArray()) letters.add(this.alphabet.get(String.valueOf(c)));  
 return letters;  
 }  
  
 // Getters and Setters  
 public List<String> getAnswers() {  
 return answers;  
 }  
 public List<String> getWords() {  
 return words;  
 }  
 public List<Letter> getAnswer() {  
 return answer;  
 }  
 public void setAnswer(String answer) {  
 if (isValidWord(answer))  
 this.answer = stringToLetterList(answer);  
 }  
 public void clearGuess() {  
 this.currentGuess = new ArrayList<>();  
 }  
 public int getTurnCount() {  
 return turnCount;  
 }  
 public boolean isSpoilerFlag() {  
 return spoilerFlag;  
 }  
 public boolean isRandomFlag() {  
 return randomFlag;  
 }  
 public List<Letter> getCurrentGuess() {  
 return currentGuess;  
 }  
 public List<Letter> getUnusedLetters() {  
 return unusedLetters;  
 }  
 public List<Letter> getWrongLetters() {  
 return wrongLetters;  
 }  
 public List<Letter> getPartialLetters() {  
 return partialLetters;  
 }  
 public List<Letter> getCorrectLetters() {  
 return correctLetters;  
 }  
 public boolean getWinFlag() { return winFlag; }  
 public boolean getGameFlag() { return gameFlag; }  
 public boolean getStrictFlag() { return strictFlag; }  
 public List<List<Letter>> getSubmittedGuesses() { return submittedGuesses; }  
}

#### Letter.java

import java.util.ArrayList;  
import java.util.List;  
  
public class Letter {  
 private final String name;  
 private List<Integer> posList; // Positions that the letter appears in the answer  
 private int buttonState;  
 private int[] roundStates;  
 private boolean isInGame;  
 private boolean hasBeenUsed;  
  
 public Letter(String name) {  
 this.name = name;  
 initialise();  
 }  
  
 public void initialise() {  
 this.buttonState = Model.UNASSIGNED;  
 this.hasBeenUsed = false;  
 this.isInGame = false;  
 this.posList = new ArrayList<>();  
 this.roundStates = new int[]{Model.WRONG, Model.WRONG, Model.WRONG, Model.WRONG, Model.WRONG};  
 }  
  
 public void setInGame() {  
 this.isInGame = true;  
 this.roundStates = new int[]{Model.PARTIAL, Model.PARTIAL, Model.PARTIAL, Model.PARTIAL, Model.PARTIAL};  
 for (int i : this.posList) this.roundStates[i] = Model.CORRECT;  
 }  
  
 public void addPos(int i) {  
 this.posList.add(i);  
 }  
  
 public String getName() {  
 return name;  
 }  
  
 public List<Integer> getPosList() {  
 return posList;  
 }  
  
 public int getButtonState() {  
 return buttonState;  
 }  
  
 public void setButtonState(int buttonState) {  
 this.buttonState = buttonState;  
 }  
  
 public boolean hasBeenUsed() {  
 return hasBeenUsed;  
 }  
  
 public void setHasBeenUsed(boolean hasBeenUsed) {  
 this.hasBeenUsed = hasBeenUsed;  
 }  
  
 public boolean isInGame() {  
 return isInGame;  
 }  
  
 public int[] getRoundStates() { return roundStates; }  
}

#### Junit.java

import org.junit.jupiter.api.DisplayName;  
import org.junit.jupiter.api.RepeatedTest;  
import org.junit.jupiter.api.Test;  
  
import java.io.IOException;  
  
import static org.junit.jupiter.api.Assertions.\*;  
  
public class JUnit {  
 Model model;  
  
 boolean invariant(Model model) {  
 return model.getTurnCount() <= Model.NUMGUESSES;  
 }  
  
 @Test  
 @DisplayName("Turn Count Updates Properly")  
 void test1() throws IOException {  
 model = new Model(1,0,0);  
 assertTrue(invariant(model));  
 assertEquals(0, model.getTurnCount());  
 model.submitWord("aaaaa");  
 model.submitGuess();  
 model.submitWord("bbbbb");  
 model.submitGuess();  
 assertEquals(2, model.getTurnCount());  
 assertTrue(invariant(model));  
 }  
  
 @Test  
 @DisplayName("Letter State Lists Update when Guess Submitted")  
 void test2() throws IOException {  
 Model.FIXED\_ANSWER = "tires";  
 model = new Model(0,0,0);  
 assertTrue(invariant(model));  
 model.submitWord("tried");  
 model.submitGuess();  
 assertEquals(21, model.getUnusedLetters().size());  
 assertEquals(2, model.getCorrectLetters().size());  
 assertEquals(1, model.getWrongLetters().size());  
 assertEquals(2, model.getPartialLetters().size());  
 assertTrue(invariant(model));  
 }  
  
 @Test  
 @DisplayName("Game is Set as Over With No Win After NUMGUESSES is Passed")  
 void test3() throws IOException {  
 model = new Model(1, 0, 0);  
 assertTrue(invariant(model));  
 for (int i = 0; i < Model.NUMGUESSES; i++) {  
 model.submitWord("aaaaa");  
 model.submitGuess();  
 }  
 assertFalse(model.getGameFlag());  
 assertFalse(model.getWinFlag());  
 assertTrue(invariant(model));  
 }  
}