Wordle Clone

Team 10

Joe Hummer, jmhummer@ncsu.edu

Ben Morris, bcmorri4@ncsu.edu

Nick Sanford, nksanfor@ncsu.edu

Nick Schauer, njschaue@ncsu.edu

NOTE: Your final documents should **NOT** contain any << >>'s or any comments or suggestions given for each section.

Introduction: **Wordle Clone**

Wordle is a simple five-letter word guessing game that rose to popularity in 2021. During its viral rise it even caught The New York Times Company eye who acquired it to challenge its millions of subscribers, and the general public, to guess the five-letter 'Wordle' of the day. Today, it is a daily routine for some to guess the word and see how long of a streak they can hold. The goal of this comprehensive exercise is to create a similar clone to the popular word guessing game.

The program will select a random word from a list of five-letter words for the player to guess. The player will then be given six tries to guess the word. After every guess, each letter is marked by green, yellow or gray: green indicating that the letter is correct and in the correct position, yellow indicating the letter is in the word but in the wrong position, and gray indicating that the letter is not in the word at all. Each five-letter guess must be a word otherwise the program will prompt the player to try again before accepting the guess and marking any letters.

This clone will be made for two players. Each player will have 5 rounds to try and guess a random five-letter word. If a player successfully guesses the word they will be awarded points based on how many guesses it took (more points for less guesses). The player with the most points at the end of the 5 rounds wins the game.

Software Requirements

Program Start

You will use the following commands to compile and run the Wordle program:

- `$javac Wordle.java`
- `$java Wordle`

Input files (*hard coded*):

- **WordleList.txt -** The list of five-letter words to randomly select the Wordle. This list has inappropriate words removed from it.
- **GuessList.txt -** The list of five-letter words to confirm a correctly spelled English language word.

Start Menu:

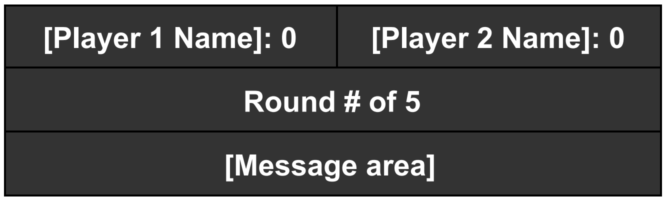
- The game will launch to the GUI below.
- Top Grid:
    - [Player 1 Name]: 0
    - [Player 2 Name]: 0
    - Round 1 of 5
    - Input player 1's name and click enter
- The guessing grid will be 'blank' with all the tiles black with no text.
- The input box will be 'blank'
- The keyboard grid will be 'blank' with all the keys light gray and the appropriate key text in white.

Player Name Input:

- When the game starts the program will first prompt the users to input their names.
- Top Grid:
    - Input Player 1's name and click enter
    - Input Player 2's name and click enter
- When complete, player 1 will start first.

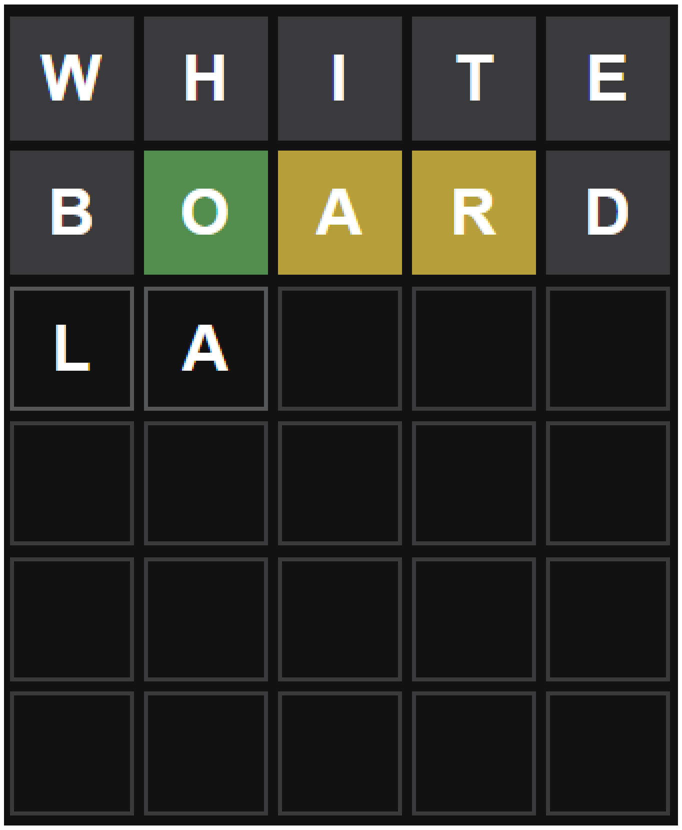
GUI / User Interface

Below is the general layout composed of 3 sections:

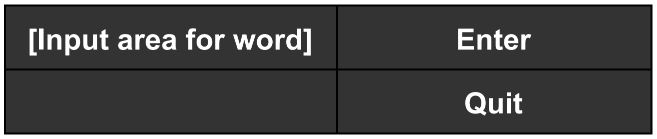


[Player 1 Name]: 0	[Player 2 Name]: 0
Round # of 5	
[Message area]	

Top Grid: Scores & Message Area



Middle Grid: Guesses



[Input area for word]	Enter
	Quit

Bottom Grid: Inputs & Quit

Gameplay/GUI

- **Top Grid: Scores & Message Area**
  - Scores
    - Will always display player 1 and 2 scores as game goes on.
  - Rounds
    - Displays the current round you are in.
  - Message Area (and or Prompts)
    - “[Player#Name]: ” - whos turn it is and a message if prompted

                - “Guess the wordle.” - Start of turn
                - “Not a word, try again.” - word check
                - “Correct! Click Enter to Continue.” - correct guess
                - “Out of guesses. Click Enter to Continue”
            - “Game over. [Player#Name] Wins!” OR “Game over. It is a TIE!”
- **Middle Grid: Guesses**
    - Shows words guessed and changes each letter’s colors based on the letter indicator.
- **Bottom Grid: Inputs & Quit**
    - 5-letter word input area (must match a 5 letter english word).
    - Enter button - move forward based on message prompt or enter letter for guess
    - Quit button - quit game

Rules

- This is a two player game.
- The game has five rounds.
- Each round, each player will guess at a Wordle.
- Each round, each player will score points based on the scoring system below.
- Each Wordle will be unique from the WordleList (no Wordle can be randomly selected more than once).
- Each player gets six guesses at their Wordle.
- Each guess must be a valid five-letter word (check against the GuessList).
- After each guess, if it is not the Wordle, the color of the tiles/letters will change to show how close your guess was to the Wordle.
    - **Green** indicates the letter is in the word and in the correct spot.
    - **Yellow** indicates the letter is in the word but in the wrong spot.
    - **Gray** indicates the letter is not in the word in any spot.
    - Letters guessed will be indicated by **dark** **gray** letter keys.
    - Letters not yet guessed will be indicated by light **gray** letter keys.

Score

If the player correctly guesses the Wordle, they get the associated points below based on the number of guesses needed. Zero points are awarded if the player is unable to guess the word in six tries. The cumulative score will be computed after each round for each player. The player at the end with the most points wins.

1. Correct guess on **first** guess:	50 pts
2. Correct guess on **second** guess:	40 pts
3. Correct guess on **third** guess:	30 pts
4. Correct guess on **fourth** guess:	20 pts
5. Correct guess on **fifth** guess:	10 pts
6. Correct guess on **sixth** guess:	0 pts

Testing Mode

Test mode can be activated by entering “-1” as a command line argument directly after the java file name (e.g. “`$java Wordle -1`”). The test mode differs from the standard mode in that it reads a separately prepared list of 10 words (titled TestWords.txt) in order instead of reading GuessWords.txt randomly. This allows predictable system testing of an entire Wordle game, including 5 words for each player. TestWords.txt will contain the following words in this order:

shape, circle, round, square, boxed, fruit, grape, sweet, berry, citrus

The Wordle GUI and program will be identical in all other aspects, such that the target words for each player in each round are as shown below:

Round	**Player 1 Target Word**	**Player 2 Target Word**
1	shape	circle
2	round	square
3	boxed	fruit
4	grape	sweet
5	berry	citrus

Software Design

WordleGUI

Static Method Headers

public static final int: GUI parameters based on layout and size. [TODO - work in progress as the GUI gets developed]

Instance Variables

private JLabel lblPlayer1: Player 1 name and score display.

private JLabel lblPlayer2: Player 2 name and score display.

private JLabel lblRound: Rounds display [Round # out of 5].

private JLabel lblMessage: User prompt and instructions to continue game play.

private JTextField txtInput: User input area.

private JButton btnEnter: Enter button to submit user input and continue game play based on lblMessage.

private JButton btnQuit: Exits program.

[TODO add more as the GUI gets developed]

Constructors

public WordleGUI(int testFlag): Builds Wordle GUI in ‘start game’ state. Labels at top to provide player, game, and instruction information. A board in the middle to display guesses and indicators per each letter. And an input area, with an enter and quit button at the bottom. The testFlag is passed to the Wordle class. This flag is used to designate if the program should load/run in ‘test mode’ but the value of -1.

Instance Methods

public void actionPerformed(ActionEvent e): When the enter button is pressed based on the lblMessage and user input the game will advance to the next stage. When the quit button is pressed the program exits immediately.

public static void main(String[] args): Starts the Wordle game by referencing the constructor to build the GUI and passing testFlag if ‘test mode’ is applicable. A try/catch is used to validate any command arguments that may be misused.

WordleGame

Static Method Headers

public static final int RANDOM_GAME = 0: Indicates that a random game should be played.

public static final int NUMBER_OF_PLAYERS = 2: Number of players in game.

public static final int NUMBER_OF_ROUNDS = 5: Number of rounds in game.

public static final int NUMBER_OF_GUESSES = 6: Number of guesses per player in a round.

public static final int NUMBER_OF_LETTERS = 5: Number of letters in a wordle.

public static final int POINTS_1 = 50: Number of points for correct guess on 1st attempt.

public static final int POINTS_2 = 40: Number of points for correct guess on 2nd attempt.

public static final int POINTS_3 = 30: Number of points for correct guess on 3rd attempt.

public static final int POINTS_4 = 20: Number of points for correct guess on 4th attempt.

public static final int POINTS_5 = 10: Number of points for correct guess on 5th attempt. With 0 points being awarded after the 5th attempt.

Instance Variables

private int currentPlayer: Keeps track of current player in game.

private int currentRound: Keeps track of current round in game.

private int currentGuess: Keeps track of current guess for player in game.

private Player[] player: Creates player objects depending on how many players.

private Wordle wordle: Creates wordle object to check user inputs against.

private Board board: Creates board object to update GUI for letters and indicators.

Constructors

public WordleGame(int testFlag): Creates wordle object and passes testFlag depending on if the ‘test mode’ is wanted.

Instance Methods

public void newGame(): Used to start a new game. Creates players and board used in game and sets current player, round and guess to 1.

public void next(String input, String oldMessage): Holds the logic to the Wordle game. Based on the user input and oldMessage being displayed the next phase of the game will advance. This method will keep track of the current player, round and guess as the game continues. A return message will be provided to prompt the player to the appropriate next step in the game.

public int scoreGuess(int currentGuess): Returns the appropriate score based on the player's currentGuess.

public int getCurrentRound(): Returns the current round of the game.

Wordle

Instance Variables

private int test: Indicates if program is in test mode or standard mode

private int index: Index of GuessList array from which to pull the target word

private String wordle:Target word for user to guess

private String[] wordleList: Stores all of the possible words that could be selected as the Wordle (the target word to be guessed), populated from a text file.

private String[] guessList: Stores all of the words that are valid guesses, populated from a text file.

Constructors

public Wordle(int test, ~~int playerNum~~) {}: Reads the .txt files to create arrays of words. Three total .txt files will be read (GuessList.txt, WordlesList.txt, and TestList.txt).

public void newAnswer() { } : Randomly assigns a target word for a Wordle game. The target word is picked randomly from the WordlesList array and assigned to the **wordle** instance variable. The given **index** on the array is then replaced with a "null" value. If the **index** value randomly chosen corresponds to a "null" value on the array, the **index** value will increase by 1 until a non-"null" value is found.

If the **test** instance variable is set to -1 the program will instead assign the **index** instance variable to 0 and **wordle** will be assigned to the first value on the TestList array. The **index** will then increase by one each time the method is called. Throw an **IllegalArgumentException** with the message "TestList array length exceeded" if the **index** value in test mode is greater than the TestList array length.

Instance methods

public String getAnswer(): Returns the target word produced by newAnswer()

public boolean isWord(): Searches the GuessList array of possible words produced by Wordle and indicates if a guess is a word

public boolean isAnswer(): Determines if the input word is the target word

public void createWordleListArray(String wordleFile): Initializes the **wordleList** array with Strings read in from either the WordleList.txt file or the TestList.txt file if testing mode is on.

public void createGuessListArray(String guessFile): Initializes the **guessList** array with Strings read in from the GuessList.txt file.

Player

Static Method Headers

public static final int STARTING_POINTS = 60: The amount of points each player receives in a new round

public static final int POINT_COST_PER_GUESS = 10: The amount of points subtracted from the players score per guess

public static final int MINIMUM_GUESSES = 1: The minimum number of guesses a player can make to win a round

public static final int MAXIMUM_GUESSES = 6: The maximum number of guesses a player can make to win a round

Instance Variables

private int score: The current player's score

private String name: The player's name

Constructors

public Player(): Sets the player name to null and score to zero

Instance Methods

public int getScore(): Returns the player's current score

public void addScore(int currentGuess): Adds the appropriate amount of points to the player's score depending on the value of **currentGuess** (60 for 1 guess, 50 for 2 guesses, etc.). **Throws** an IllegalArgumentException if the number of guesses is greater than 6 or less than 1.

public String getName(): Returns the Player's name

public void addName(): Assigns or replaces the player's name

Board

Instance Variables

private char[][] letters: A 2D char array representing all the guesses made so far. Each row represents the word guessed, and each column within the row is a char representing the letter in that position in the word.

private String[][] colors: A 2D String array representing all the colors associated with each guess made so far. Each row contains a String array where each element is the String of the color associated with that particular letter. These color Strings can be "green" to indicate the letter is in the correct position, "yellow" to indicate that the letter is in the word, but is in the incorrect position, or "gray" to indicate that the letter does not exist in the word.

private int currentGuesses: The number of guesses made so far and currently stored in the Board object. This is also used as the index to store the next guess into the letters and colors arrays.

Constructors

public Board (int numTotalGuesses, int wordLength): Creates a new **Board** object by initializing the **letters** and **colors** arrays to both have a number of rows equal to the total number of guesses that a player can make, and a number of columns equal to the length of each word.

Instance Methods

public void addGuess(char[] guessLetters, String[] guessColors): Adds an array of a word, called guessLetters, to the 2D **letters** array, and adds an array of colors, called guessColors, to the 2D **colors** array. Throws an **IllegalArgumentException** if the length of the referenced arrays is greater than the length of the 2D arrays' row length. Note that the elements in the passed-in arrays are copied by value into the **letters** and **colors** arrays.

public char[] getGuessLetters(int index): Returns a reference to the char array at the specified index within the **letters** array.

public String[] getGuessColors(int index): Returns a reference to the String array at the specified index within the **colors** array.

public String toString(): Returns a String representation of the **Board** object. For each guess stored in the **letters** array/**colors** array, two lines are added to the String. The first line contains the word stored in the letters array for that row, while the second line contains the colors for each letter, separated by a space. For instance, if the **letters** array and **colors** array only have one guess stored, the output might be: happy\ngreen yellow green gray gray \n

Implementation

Object-Oriented Programming will be used to divide the Wordle Clone into different objects encapsulating data and performing tasks. The WordleGame (client) is used to execute the overall logic of gameplay while calling on the Wordle, Player, and Board classes. Wordle is used to store, change and test if a player has guessed the Wordle. The Player class is used to store the player's name and score. While the Board class is used to store the letter and indicator based on the 'taken' player guesses. The WordleGUI is used to provide the interactive user interface during the gameplay calling on the WordleGame to update and display the current game results and prompts.

Other concepts being used:

- WordleGUI
  - [TODO - list appropriate packages used to for GUI]
  - The program must exit (stop) as soon as the user clicks the Quit button, `System.exit(1);`.
- WordleGame
  - [nothing special at the moment]
- Wordle
  - Use the `Random()` class to randomly select a new wordle in the **newAnswer()** method.
  - Use a `Scanner()` to input the WordleList.txt, GuessList.txt, and TestWords.txt files into individual arrays in their individual **Create()** methods. Scan each word using line-based processing, `Scanner lineScanner = new Scanner(line);`.
- Player
  - Use Illegal argument exception when the amount of guesses is out of bounds
  - Use a For loop to determine the number of points a player receives in a round
  - Use Getters and Setters for the player name and score
- Board
  - [nothing special at the moment]

Testing

System Testing

1. The System testing of the Wordle Clone program should be performed according to the SystemTestPlan_CE.pdf document.
2. List any test files required to run these tests.
3. Discuss any special testing scenario requirements (for example, test arrays with prespecified values to be used instead of random numbers while in test mode, command line arguments to run in test mode, etc).

Unit Testing

- No unit test files or documentation are required this semester, so this can be omitted.

Team Reflection

- Challenges faced/Lessons learned
    - What did the team learn during the project development process?
    - What problems arose and how did you solve them?

In addition to above team reflection that must be included in this document each team member must also complete an individual reflection (not in this document).

The following information should be included in each individual's peer review form of themselves.

- What did you like/dislike about the exercise?
- Any suggested changes/improvements?
- Add any comments/thoughts you care to share.