Learning Objectives

Learners will be able to...

- **Create a Wordle solver**
- **Use linear search to reduce set of possible answers**
- **Add project to GitHub portfolio**

Wordle

Wordle Solver

Congratulations on making it this far in your data structures and algorithms course! This shows that you are well-invested in computer science. One of the best ways to improve your skills and test your abilities is to work on side projects.

In this course, we have covered a lot of material that can be applied to a variety of projects. One project that we could work on is a Wordle solver. Wordle is a web-based word game created by Josh Wardle. It was launched in October 2021 and quickly became a global sensation. A Wordle solver would use data structures and algorithms to help players guess the correct word in as few tries as possible.

Once we have completed the Wordle solver, we can upload it to GitHub to start building our project portfolio. This will be a great way to showcase our skills and experience to potential employers.

Playing Wordle

To play Wordle, you have six chances to guess a five-letter word. After each guess, the game will give you feedback on how close you are to the correct word. The feedback is color-coded:

- **Green:** The letter is in the correct position.
- **Yellow:** The letter is in the word, but not in the correct position.
- **Gray:** The letter is not in the word.

You can use this feedback to inform your next guess and narrow down the possible solutions. The goal is to guess the correct word in as few tries as possible. Wordle is a simple game, but it can be surprisingly challenging. It's a great way to improve your vocabulary and test your logic skills. And it's a lot of fun!

Here are some tips for playing Wordle:

- Start with a word that contains a lot of common letters, such as "a", "e", "i", "o", "s", "t", "r", or "l".
- Pay attention to the feedback after each guess. If a letter is green, you

know it's in the correct position. If it's yellow, you know it's in the word, but not in the correct position. And if it's gray, you know it's not in the word at all.
- Use your knowledge of word prefixes, suffixes, and common word roots to help you guess the correct word.
- Don't be afraid to guess the same word multiple times. If you're stuck, it's sometimes helpful to guess a word that you've already guessed, just to see if it's in the correct position.

With a little practice, you'll be solving Wordle puzzles in no time. But with a little computer science you can solve all the time XD.

The solver asks you to input information (see text below) based on the color-coded feedback from the game. The solver will return a list of possible solutions based on the criteria you provide. After each guess in the game, update the criteria to get a more refined list of potential solutions. It is okay if you do not fully understand how to use the solver. We will go over this in more detail.

```
Your current cell is marked with a '='.
 * Enter a letter if you know it belongs in that cell.
 * Enter a space if you want to skip to the next cell.
 * Enter '1' to move one cell back and make changes.
 * Enter '0' to exit the board editing process.
[=, _, _, _, _]
```

Use the `TRY IT` button below to try a sample of the code we are going to create. See if you can use the program to solve today's Wordle.

GitHub

As you work on personal projects, it is good to save them somewhere. One of those places to save them could be a VCS (Version Control System) like GitHub. GitHub is particularly useful because it can help you maintain your code, work history, and share it with others. Sharing it with others, is one the key things that can help you land a job down the line. Sharing your portfolio, can be done by giving links to your GitHub or your own personal website. For now, lets focus on the GitHub.

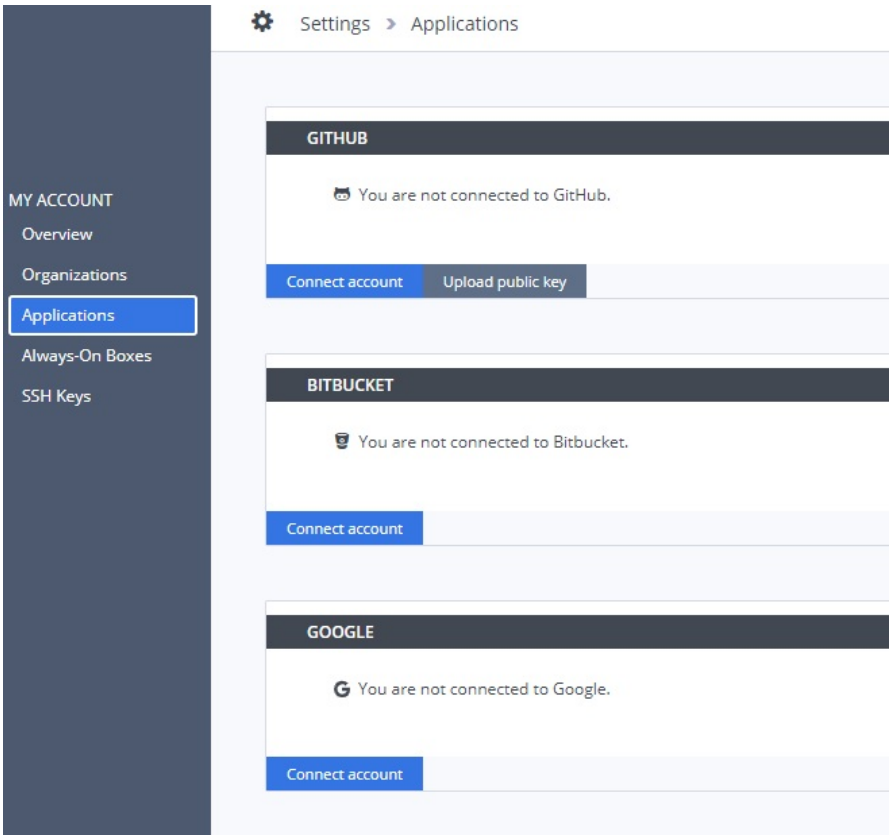
Connecting to GitHub

In order to showcase this project for your portfolio, you are going to use GitHub. If you do not yet have an account, please create one now. We are going to clone a repo that will contain the code for your Wordle solver.

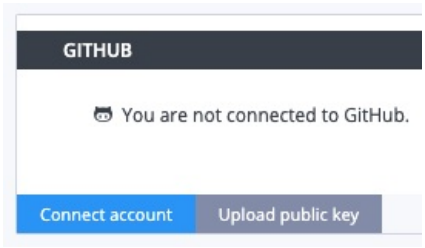
Connecting GitHub and Codio

You need to connect GitHub to your Codio account. This only needs to be done one time.

- In your Codio account, click on your username
- Click on **Applications**



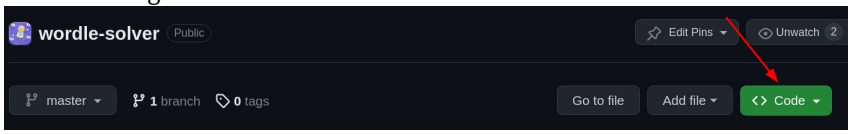
- Under GitHub, click on **Connect account**



- You will be using an SSH connection, so you need to click on **Upload public key**

Repo SSH Information

- Go to the wordle-solver repo. This repo is the starting point for your project.
- Click on the green “Code” button.



- Copy the **SSH** information. It should look something like this:

```
git@github.com:codio-content/wordle-solver.git
```

info

Important

If you do not use the SSH information, you will have to provide your username and Personal Access Token (PAT) to GitHub each time you push or pull from the repository. See this documentation for setting up a PAT for your GitHub account.

In the Terminal

- Clone the repo. Your command should look something like this:

```
git clone git@github.com:codio-content/wordle-solver.git
```

- You should see a `wordle-solver` directory appear in the file tree.
- In that tree you should see the `WordleSolver.java`, `words.txt`, and `README.md` files.
- When asked, you should type yes

On the last page, we'll show you how to push to GitHub. You are now ready to create your first project in your portfolio.

Goal

Wordle, Searching, and Sorting

Using our knowledge of arrays and searching algorithms, we will create an interactive Wordle solver. After each guess, we will use the feedback to help our code narrow down the search space.

- If a letter is green, we know it is in the correct position. Therefore, we will ensure that all of the remaining words in the search space have that letter in the same location.
- If a letter is yellow, we know it is in the word, but not in the correct position. In this case, we will create a criteria that ensures that the yellow letter appears in at least one of the remaining words.
- If a letter is gray, we know it is not in the word at all. Therefore, we will remove all words from the search space that contain that letter.

By iteratively narrowing down the search space in this way, our code will be able to solve Wordle puzzles in a relatively short amount of time.

To help with our search, a file `words.txt`. is on your left panel, which is all the 5 letter words. **Please do not modify it**. We will use it as our repertoire of words and narrow down our search from there. Alright, let's get started!

▼ **Possible Words**
If you are interested in the possible five-letter words are game can choose from, click the button below to open the file. This will open a file in a new tab in Codio. You will need to switch back to the Guide tab in order to continue with the assignment.

Intro

Welcome to this coding guide! We're going to build a Wordle Solver in Java. Wordle is a word puzzle game that challenges players to guess a five-letter word within six attempts. Our program will help to find the possible words that fit the criteria you input, making it easier to win the game.

On This Page...

We will follow an incremental approach to understand and build the Wordle Solver. Here's a quick look at what our Java program will do:

1. **Initialize an Empty Board**: Represent the puzzle board where you input known and unknown letters.
2. **Load a Dictionary**: Read from a file called `words.txt` that contains potential five-letter words.
3. **Get User Input**: Allow the user to specify known letters, ignored letters, and mandatory letters.
4. **Search for Matches**: Find all words that meet the specified criteria.
5. **Display Results**: Show the found words to the user.

Getting started

Let's start by looking at some essential building blocks of our program. First off, we need to import Java's utility libraries for I/O and data structures.

```
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.util.List;
import java.util.Scanner;
import java.util.ArrayList;
import java.util.Arrays;
```

Here, we import classes for reading from a file (`BufferedReader`, `FileReader`, `IOException`), data structure manipulation (`List`, `ArrayList`, `Arrays`), and for user input (`Scanner`).

Next, we declare our main class `WordleSolver`. Before we dive into the methods, we define a couple of global variables that we'll use throughout the program:

```
public class WordleSolver {
    private static List<String> ignoreWords = new ArrayList<>();
    private static List<String> board = new ArrayList<>();

}
```

`ignoreWords`: A list that will hold any letters the user wishes to ignore when searching for potential words.

- `board`: This list represents the board state, showing which letters are known and where there are gaps.

We'll learn how to use these variables as we proceed further in this guide.

▼ **Code**
Your code should look like this:

```
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.util.List;
import java.util.Scanner;
import java.util.ArrayList;
import java.util.Arrays;

public class WordleSolver {
    private static List<String> ignoreWords = new ArrayList<>();
    private static List<String> board = new ArrayList<>();

}
```

The Board

Now that we have a foundation on which to build, we will focus on creating and managing the game board.

On This Page...

- **The Board**: Understand how the board represents the state of the game.
- **Reset the Board**: Initialize or reset the board to its original state.
- **Display the Board**: Show the current board state to the user.

The Board

In Wordle, the board is a simple 5x1 grid where each cell can contain a letter of the alphabet or an underscore (empty). We represent this board (referred to as `board` in our program) as a list of strings. For example, if the board state is `["_", "_", "a", "_", "_"]` that means that the third letter of the target word is known to be `a`, while the rest is unknown.

The first thing we need to do display the contents of the board. Create the `display` method that takes a list of strings as a parameter. The method does not return a value. Because our board uses the `List` interface, using a simple print statement will print the list in a human readable way.

```
    private static void display(List<String> currentBoard) {
        System.out.println(currentBoard);
    }
```

However, if you create a `List` object without a value, Java defaults to an empty list. So running our program now will not result in any visual output. Let's create the `resetBoard` method that sets the value of each string to an underscore. The `resetBoard` method has no parameters and does not return anything. Remember, `board` is a global variable.

```
    private static void resetBoard() {
        board.clear();
        for (int i = 0; i < 5; i++) {
            board.add("_");
        }
    }
```

Our program is controlled via the `play` method. For now, we are going to create a very simple method that resets the board and then displays it.

```
    public static void play() {
        resetBoard();
        display(board);
    }
```

Finally, let's add a `main` method where we call the `play` method.

```
    public static void main(String[] args) {
      play();
    }
```

When you compile and run your code, you should see the following output:

```
[_, _, _, _, _]
```

Did you see the five underscores representing the board? If everything looks good, we can move on to the next part, where we start building more interactive features.

▼ Code
Your code should look like this:

```java
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.util.List;
import java.util.Scanner;
import java.util.ArrayList;
import java.util.Arrays;

public class WordleSolver {
    private static List<String> ignoreWords = new ArrayList<>();
    private static List<String> board = new ArrayList<>();

    private static void display(List<String> currentBoard) {
        System.out.println(currentBoard);
    }

    private static void resetBoard() {
        board.clear();
        for (int i = 0; i < 5; i++) {
            board.add("_");
        }
    }

    public static void play() {
        resetBoard();
        display(board);
    }

    public static void main(String[] args) {
      play();
    }
}
```

User Input

We discussed creating and managing the game board. We briefly touched upon the `initializeBoard()` function, which calls another function, `userEntry(int location)`, to allow the user to interact with the board. In this page, we will delve into the `userEntry()` function and understand how it enables the user to update the board.

On This Page...

- **Read from the Terminal**: Capture text that the user inputs in the terminal.
- **Update the Board**: Use the captured input to update the state of the board.

Read from the Terminal

Users will interact with the program via the terminal. Let's create the `getInput` method that takes a string that represents a prompt and returns a string. First, print the prompt that was passed to the method. Then create a `Scanner` object to read from the terminal. Return the text that the user entered.

```
    private static String getInput(String prompt) {
        System.out.print(prompt + " ");
        Scanner scanner = new Scanner(System.in);
        return scanner.nextLine();
    }
```

To test our code, update the `play` method to print the result from the `getInput` method. Basically, the program should echo the text you enter in the terminal.

```
    public static void play() {
        System.out.println(getInput("Enter some text: "));
    }
```

Compile and run the program. What ever text you enter should be returned to you.

Update the Board

The `userEntry(int location)` method allows the user to interact with the board cell-by-cell, letting them specify a known letter, or leave it as unknown (underscore). The `location` parameter specifies which cell of the board the method is currently dealing with.

The user will be prompted to input one of the following:

- A letter if they know what letter belongs in the current cell.
- Space or an empty entry to skip to the next cell without changing it.
- `0` to exit the board editing process.
- `1` to move one cell back to make a change.

```
    private static int userEntry(int location) {
        String initialState = board.get(location);
        board.set(location, "=");

        String userInput = getInput(board.toString());

        if (userInput.equals(" ") || userInput.isEmpty()) {
            board.set(location, initialState);
            return location + 1;
        } else if (userInput.equals("0")) {
            board.set(location, initialState);
            return 5;
        } else if (userInput.equals("1")) {
            board.set(location, "_");
            return Math.max(location - 1, 0);
        } else {
            board.set(location, userInput);
            return location + 1;
        }
    }
```

Here is how the `userEntry` method works:

1. **Save the Initial State**: The initial state of the current cell is stored.
2. **Prompt User**: The current board state is displayed with a = sign at the location that the user is updating.
3. **User Input**: Calls the `getInput()` function to get the user's input.
4. **Handle Input**: Depending on the user input, the function performs one of the following actions:
   - Leave the cell unchanged and move to the next (`" "` or empty input).
   - Exit the board editing (`"0"`).
   - Move back one cell (`"1"`).
   - Update the cell with the new letter.
5. **Return Value**: Returns the index of the next cell to be edited, or `5` to indicate the user has exited the board editing process.

Next, we are going to create the `initializeBoard` method. It does not take any parameters and does not return a value. This method allows the user to continue modifying their board until they enter `0`. After which, the method displays the board to the user.

```
    private static void initializeBoard() {
        int i = 0;
        while(i < 5) {
            i = userEntry(i);
        }
        display(board);
    }
```

Before we can test our code, we need to update the `play` method. We want to give the user some brief instructions on how to interact with the program. Be sure to reset the board, then call the `initializeBoard` method.

```
    public static void play() {
        String prompt = "Your current cell is marked with a
        '='.\n * Enter a letter if you know it belongs in that
        cell.\n * Enter a space if you want to skip to the next
        cell.\n * Enter '1' to move one cell back and make
        changes.\n * Enter '0' to exit the board editing
        process.";
        System.out.println(prompt);

        resetBoard();
        initializeBoard();
    }
```

Enter the letters `d` and `r`. Then enter `1` to go back one cell and replace `r` with `c`. Finally, enter `0` to quit. Your output should look like this:

```
Your current cell is marked with a '='.
 * Enter a letter if you know it belongs in that cell.
 * Enter a space if you want to skip to the next cell.
 * Enter '1' to move one cell back and make changes.
 * Enter '0' to exit the board editing process.
[=, _, _, _, _] d
[d, =, _, _, _] r
[d, r, =, _, _] 1
[d, =, _, _, _] c
[d, c, =, _, _] 0
[d, c, _, _, _]
```

On this page, we dived deep into the `userEntry()` function, which serves as the primary interaction point for the user to update the board. We discussed the types of input the user can provide and how the function handles them to update the board accordingly.

▼ Code

Your code should look like this:

```
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.util.List;
```

```java
import java.util.Scanner;
import java.util.ArrayList;
import java.util.Arrays;

public class WordleSolver {
    private static List<String> ignoreWords = new ArrayList<>();
    private static List<String> board = new ArrayList<>();

    private static void initializeBoard() {
        int i = 0;
        while(i < 5) {
            i = userEntry(i);
        }
        display(board);
    }

    private static int userEntry(int location) {
        String initialState = board.get(location);
        board.set(location, "=");

        String userInput = getInput(board.toString());

        if (userInput.equals(" ") || userInput.isEmpty()) {
            board.set(location, initialState);
            return location + 1;
        } else if (userInput.equals("0")) {
            board.set(location, initialState);
            return 5;
        } else if (userInput.equals("1")) {
            board.set(location, "_");
            return Math.max(location - 1, 0);
        } else {
            board.set(location, userInput);
            return location + 1;
        }
    }

    private static String getInput(String prompt) {
        System.out.print(prompt + " ");
        Scanner scanner = new Scanner(System.in);
        return scanner.nextLine();
    }

    private static void display(List<String> currentBoard) {
        System.out.println(currentBoard);
    }

    private static void resetBoard() {
        board.clear();
        for (int i = 0; i < 5; i++) {
            board.add("_");
        }
    }

    public static void play() {
```

```
        String prompt = "Your current cell is marked with a
        '='.\n * Enter a letter if you know it belongs in that
        cell.\n * Enter a space if you want to skip to the next
        cell.\n * Enter '1' to move one cell back and make
        changes.\n * Enter '0' to exit the board editing
        process.";
        System.out.println(prompt);

        resetBoard();
        initializeBoard();
    }

    public static void main(String[] args) {
      play();
    }
}
```

File Handling

We delved into how the `userEntry()` function allows the user to interact with and update the game board. We learned how different user inputs are handled to update the board accordingly.

Now, to find potential matching words, our Wordle solver needs a list of possible five-letter words. These words are stored in a file named `words.txt`, where each line represents a distinct word.

On This Page...

- **Read a File**: Brief overview of how file reading works in Java.
- **Error Handling**: What happens when things go wrong.
- **Load the Dictionary**: Read the list of possible words from a file.

File Handling in Java

The Java standard library provides multiple ways to read a file. Here, we use a `BufferedReader` object because it offers a convenient `readLine()` method, which reads a line from the file and moves the cursor to the beginning of the next line.

The code employs a `try`-with-resources statement to ensure that the file is properly closed after reading, even if an exception occurs. If any `IOException` occurs (like the file not being found), the stack trace is printed to help debug the issue.

Load the Dictionary

The function `letterWords()` reads this file, populates a list of strings with its content, and returns the list. Let's look at its code:

```
    private static List<String> letterWords() {
        List<String> rows = new ArrayList<>();
        try (BufferedReader reader = new BufferedReader(new
        FileReader("wordle-solver/words.txt"))) {
            String line;
            while ((line = reader.readLine()) != null) {
                rows.add(line.trim());
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
        return rows;
    }
```

Here is how the `letterWords` method works:

1. **Initialize an Empty List**: A new ArrayList `rows` is created to hold the words.
2. **Open File**: Uses Java's `BufferedReader` to open `words.txt`.
3. **Read Lines**: Reads each line of the file, trims any whitespace, and adds it to `rows`.
4. **Error Handling**: Catches any IOException and prints the stack trace.
5. **Return the Words**: Returns the ArrayList `rows` which has all of the words from the dictionary.

Before we can test our code, we need to modify the `play` method. The entire list of words is quite long. Instead, we are going to print a list with the first five words, and another list with the last five words. Note, the dictionary of words is not in alphabetical order.

```
    public static void play() {
        List<String> words = new ArrayList<>();
        words = letterWords();
        System.out.println(words.subList(0, 5));
        System.out.println(words.subList(words.size() - 6,
        words.size() - 1));
    }
```

You should see the following output:

```
[which, there, their, about, would]
[spumy, osier, roble, rumba, biffy]
```

This loaded list is crucial for the next steps, where we will filter and narrow down the list of words based on the board state and user inputs.

▼ Code

Your code should look like this:

```
import java.io.BufferedReader;
import java.io.FileReader;
```

```java
import java.io.IOException;
import java.util.List;
import java.util.Scanner;
import java.util.ArrayList;
import java.util.Arrays;

public class WordleSolver {
    private static List<String> ignoreWords = new ArrayList<>();
    private static List<String> board = new ArrayList<>();

    private static List<String> letterWords() {
        List<String> rows = new ArrayList<>();
        try (BufferedReader reader = new BufferedReader(new
        FileReader("wordle-solver/words.txt"))) {
            String line;
            while ((line = reader.readLine()) != null) {
                rows.add(line.trim());
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
        return rows;
    }

    private static void initializeBoard() {
        int i = 0;
        while(i < 5) {
            i = userEntry(i);
        }
        display(board);
    }

    private static int userEntry(int location) {
        String initialState = board.get(location);
        board.set(location, "=");

        String userInput = getInput(board.toString());

        if (userInput.equals(" ") || userInput.isEmpty()) {
            board.set(location, initialState);
            return location + 1;
        } else if (userInput.equals("0")) {
            board.set(location, initialState);
            return 5;
        } else if (userInput.equals("1")) {
            board.set(location, "_");
            return Math.max(location - 1, 0);
        } else {
            board.set(location, userInput);
            return location + 1;
        }
    }

    private static String getInput(String prompt) {
        System.out.print(prompt + " ");
        Scanner scanner = new Scanner(System.in);
        return scanner.nextLine();
    }
```

```
    private static void display(List<String> currentBoard) {
        System.out.println(currentBoard);
    }

    private static void resetBoard() {
        board.clear();
        for (int i = 0; i < 5; i++) {
            board.add("_");
        }
    }

    public static void play() {
        List<String> words = new ArrayList<>();
        words = letterWords();
        System.out.println(words.subList(0, 5));
        System.out.println(words.subList(words.size() - 6,
        words.size() - 1));
    }

    public static void main(String[] args) {
      play();
    }
}
```

Word Data from a File

So far we have a board whose state can be edited by the user and a dictionary of words. We are now going to begin the process of using the state of the board to filter the list of words.

On This Page...

- **Data Structure Choices**: Explain why we use certain data structures.
- **Filter the List**: Reduce the number of possible words down to a smaller list.

Data Structure Choices

We use Java's `ArrayList` for both the word list (`bigList`) and the board state (`target`) because we need fast, indexed access to individual elements. In addition, we do not know how many words will match the `target`, so we need a dynamic data structure like an ArrayList.

Filter the List

The `matcher()` function is responsible for filtering the list of possible words based on the current state of the board. It takes in two lists as parameters:

- `bigList`: This is the list of all possible words (generally loaded from `words.txt` by `letterWords()`).
- `target`: This is the current state of the board, where known letters are filled in and unknowns are represented by underscores (_).

The function iterates through each word in `bigList` and tries to match it against the `target` board. If a word matches the known letters and positions on the board, it is added to a new list, `results`, which is then returned.

```
    private static List<String> matcher(List<String> bigList,
        List<String> target) {
        List<String> results = new ArrayList<>();

        for (String word : bigList) {
            boolean isMatch = true;
            for (int i = 0; i < 5; i++) {
                if (!target.get(i).equals("_") &&
    !target.get(i).equals(String.valueOf(word.charAt(i)))) {
                    isMatch = false;
                    break;
                }
            }
            if (isMatch) {
                results.add(word);
            }
        }

        return results;
    }
```

Here is how the `matcher` method works:

1. **Initialize `results`**: Create a new list to hold the matching words.
2. **Iterate Over Words**: Goes through each word in `bigList`.
3. **Inner Loop**: Checks each character of the current word against the corresponding character in `target`.
4. **Add to `results`**: If a word matches the board state, it's added to `results`.
5. **Return `results`**: Return the list of matching words.

Before we can test our code, we need to modify the `play` method. Add back the instructions and the board. Read from the dictionary of words, and then match them with the state of the board. Then print out the matching words.

```
    public static void play() {
        String prompt = "Your current cell is marked with a
        '='.\n * Enter a letter if you know it belongs in that
        cell.\n * Enter a space if you want to skip to the next
        cell.\n * Enter '1' to move one cell back and make
        changes.\n * Enter '0' to exit the board editing
        process.";
        System.out.println(prompt);

        resetBoard();
        initializeBoard();

        List<String> rows = letterWords();
        List<String> results = matcher(rows, board);

        System.out.println("Here are the results of the
        matches:");
        System.out.println(results);
    }
```

Compile and run the program. Enter the letters `r`, `a`, and `n` into the board. Then exit the editing by entering `0`. You should see the following output:

```
Your current cell is marked with a '='.
 * Enter a letter if you know it belongs in that cell.
 * Enter a space if you want to skip to the next cell.
 * Enter '1' to move one cell back and make changes.
 * Enter '0' to exit the board editing process.
[=, _, _, _, _] r
[r, =, _, _, _] a
[r, a, =, _, _] n
[r, a, n, =, _] 0
[r, a, n, _, _]
Here are the results of the matches:
[range, ranch, ranks, rangy, randy, rants, rands]
```

This is a good first start to filtering words, but Wordle gives us more information that can further help us reduce our list of possible words.

▼ Code
Your code should look like this:

```
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.util.List;
import java.util.Scanner;
import java.util.ArrayList;
import java.util.Arrays;

public class WordleSolver {
    private static List<String> ignoreWords = new ArrayList<>();
    private static List<String> board = new ArrayList<>();

    private static List<String> matcher(List<String> bigList,
        List<String> target) {
        List<String> results = new ArrayList<>();

        for (String word : bigList) {
            boolean isMatch = true;
            for (int i = 0; i < 5; i++) {
                if (!target.get(i).equals("_") &&
        !target.get(i).equals(String.valueOf(word.charAt(i)))) {
                    isMatch = false;
                    break;
                }
            }
            if (isMatch) {
                results.add(word);
            }
        }

        return results;
    }
```

```
    private static List<String> letterWords() {
        List<String> rows = new ArrayList<>();
        try (BufferedReader reader = new BufferedReader(new
        FileReader("wordle-solver/words.txt"))) {
            String line;
            while ((line = reader.readLine()) != null) {
                rows.add(line.trim());
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
        return rows;
    }

    private static void initializeBoard() {
        int i = 0;
        while(i < 5) {
            i = userEntry(i);
        }
        display(board);
    }

    private static int userEntry(int location) {
        String initialState = board.get(location);
        board.set(location, "=");

        String userInput = getInput(board.toString());

        if (userInput.equals(" ") || userInput.isEmpty()) {
            board.set(location, initialState);
            return location + 1;
        } else if (userInput.equals("0")) {
            board.set(location, initialState);
            return 5;
        } else if (userInput.equals("1")) {
            board.set(location, "_");
            return Math.max(location - 1, 0);
        } else {
            board.set(location, userInput);
            return location + 1;
        }
    }

    private static String getInput(String prompt) {
        System.out.print(prompt + " ");
        Scanner scanner = new Scanner(System.in);
        return scanner.nextLine();
    }

    private static void display(List<String> currentBoard) {
        System.out.println(currentBoard);
    }

    private static void resetBoard() {
        board.clear();
        for (int i = 0; i < 5; i++) {
            board.add("_");
        }
```

```
    }

    public static void play() {
        String prompt = "Your current cell is marked with a
        '='.\n * Enter a letter if you know it belongs in that
        cell.\n * Enter a space if you want to skip to the next
        cell.\n * Enter '1' to move one cell back and make
        changes.\n * Enter '0' to exit the board editing
        process.";
        System.out.println(prompt);

        resetBoard();
        initializeBoard();

        List<String> rows = letterWords();
        List<String> results = matcher(rows, board);

        System.out.println("Here are the results of the
        matches:");
        System.out.println(results);
    }

    public static void main(String[] args) {
      play();
    }
}
```

Refining the Search

The Wordle game often involves strategic plays where you might want to avoid certain letters or make sure some are included in your guesses. We are going to introduce the `remover()` and `mustHave()` methods as additional layers of filtering for the possible words. These methods make use of the feedback that Worlde gives us each time we make a guess.

On This Page...

Code Breakdown

1. **Initialize a New List**: In both methods, a new list is initialized to hold the filtered words.
2. **Iterate Through Words**: Each method iterates over every word in the `bigList`.
3. **Inner Loop for Conditions**: Nested loops check each word against the conditions specified by the `ignore` or `must` strings.
4. **Add Filtered Words**: Words meeting the conditions are added to the new list.
5. **Return Lists**: Return the list of words that meet the specified criteria.

Removing Words

The `remover` method takes a list of strings representing all of the possible words and a string containing letters we want to ignore. The method returns a list of strings.

```
    private static List<String> remover(List<String> bigList,
        String ignore) {
        List<String> newList = new ArrayList<>();
        for (String word : bigList) {
            boolean flag = true;
            for (char ch : ignore.toCharArray()) {
                if (word.indexOf(ch) != -1) {
                    flag = false;
                    break;
                }
            }
            if (flag) {
                newList.add(word);
            }
        }
        return newList;
    }
```

Before we can test our code, we need to update the `play` method. After reading the dictionary of words, prompt the user to provide any letters to ignore. Update the possible list of words by removing words that contain letters that should be ignored. Now match the updated list to the state of the board, and then print the list.

```
    public static void play() {
        String prompt = "Your current cell is marked with a
        '='.\n * Enter a letter if you know it belongs in that
        cell.\n * Enter a space if you want to skip to the next
        cell.\n * Enter '1' to move one cell back and make
        changes.\n * Enter '0' to exit the board editing
        process.";
        System.out.println(prompt);

        resetBoard();
        initializeBoard();

        List<String> rows = letterWords();
        String ignore = getInput("Type in the letters to ignore
        (if any): ");
        rows = remover(rows, ignore);
        List<String> results = matcher(rows, board);

        System.out.println("Here are the results of the
        matches:");
        System.out.println(results);
    }
```

Just as before, we are going to enter the letters `r`, `a`, and `n` into the board. We are then going to enter `y` for the letter to ignore. You should see the following output:

```
Your current cell is marked with a '='.
 * Enter a letter if you know it belongs in that cell.
 * Enter a space if you want to skip to the next cell.
 * Enter '1' to move one cell back and make changes.
 * Enter '0' to exit the board editing process.
[=, _, _, _, _] r
[r, =, _, _, _] a
[r, a, =, _, _] n
[r, a, n, =, _] 0
[r, a, n, _, _]
Type in the letters to ignore (if any):  y
Here are the results of the matches:
[range, ranch, ranks, rants, rands]
```

Required Letters

Similar to `remover`, the `mustHave` method also filters the list of words but in the opposite way. This method also takes a list of strings and a string, and returns a list of strings. It ensures that the returned list only includes words containing certain required letters.

```
    private static List<String> mustHave(List<String> bigList,
        String must) {
        List<String> newList = new ArrayList<>();
        for (String word : bigList) {
            boolean flag = true;
            for (char ch : must.toCharArray()) {
                if (word.indexOf(ch) == -1) {
                    flag = false;
                    break;
                }
            }
            if (flag) {
                newList.add(word);
            }
        }
        return newList;
    }
```

Before we can test our code, we need to update the `play` method. After reading the dictionary of words, match them to the state of the board. Prompt the user to provide any letters that are required. Now filter the list against the required letters. Print the list.

```
    public static void play() {
        String prompt = "Your current cell is marked with a
        '='.\n * Enter a letter if you know it belongs in that
        cell.\n * Enter a space if you want to skip to the next
        cell.\n * Enter '1' to move one cell back and make
        changes.\n * Enter '0' to exit the board editing
        process.";
        System.out.println(prompt);

        resetBoard();
        initializeBoard();

        List<String> rows = letterWords();
        List<String> results = matcher(rows, board);
        String must = getInput("Type in the letters to include
        but not sure of location: ");
        results = mustHave(results, must);

        System.out.println("Here are the results of the
        matches:");
        System.out.println(results);
    }
```

Enter once more the letters `r`, `a`, and `n` into the board. Then tell the program the list of words must contain an `s`. You should see the following output:

```
Your current cell is marked with a '='.
 * Enter a letter if you know it belongs in that cell.
 * Enter a space if you want to skip to the next cell.
 * Enter '1' to move one cell back and make changes.
 * Enter '0' to exit the board editing process.
[=, _, _, _, _] r
[r, =, _, _, _] a
[r, a, =, _, _] n
[r, a, n, =, _] 0
[r, a, n, _, _]
Type in the letters to include but not sure of location:  s
Here are the results of the matches:
[ranks, rants, rands]
```

▼ Code

Your code should look like this:

```
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.util.List;
import java.util.Scanner;
import java.util.ArrayList;
import java.util.Arrays;

public class WordleSolver {
    private static List<String> ignoreWords = new ArrayList<>();
    private static List<String> board = new ArrayList<>();
```

```
    private static List<String> mustHave(List<String> bigList,
        String must) {
        List<String> newList = new ArrayList<>();
        for (String word : bigList) {
            boolean flag = true;
            for (char ch : must.toCharArray()) {
                if (word.indexOf(ch) == -1) {
                    flag = false;
                    break;
                }
            }
            if (flag) {
                newList.add(word);
            }
        }
        return newList;
    }

    private static List<String> remover(List<String> bigList,
        String ignore) {
        List<String> newList = new ArrayList<>();
        for (String word : bigList) {
            boolean flag = true;
            for (char ch : ignore.toCharArray()) {
                if (word.indexOf(ch) != -1) {
                    flag = false;
                    break;
                }
            }
            if (flag) {
                newList.add(word);
            }
        }
        return newList;
    }

    private static List<String> matcher(List<String> bigList,
        List<String> target) {
        List<String> results = new ArrayList<>();

        for (String word : bigList) {
            boolean isMatch = true;
            for (int i = 0; i < 5; i++) {
                if (!target.get(i).equals("_") &&
        !target.get(i).equals(String.valueOf(word.charAt(i)))) {
                    isMatch = false;
                    break;
                }
            }
            if (isMatch) {
                results.add(word);
            }
        }

        return results;
    }

    private static List<String> letterWords() {
```

```
        List<String> rows = new ArrayList<>();
        try (BufferedReader reader = new BufferedReader(new
        FileReader("wordle-solver/words.txt"))) {
            String line;
            while ((line = reader.readLine()) != null) {
                rows.add(line.trim());
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
        return rows;
    }

    private static void initializeBoard() {
        int i = 0;
        while(i < 5) {
            i = userEntry(i);
        }
        display(board);
    }

    private static int userEntry(int location) {
        String initialState = board.get(location);
        board.set(location, "=");

        String userInput = getInput(board.toString());

        if (userInput.equals(" ") || userInput.isEmpty()) {
            board.set(location, initialState);
            return location + 1;
        } else if (userInput.equals("0")) {
            board.set(location, initialState);
            return 5;
        } else if (userInput.equals("1")) {
            board.set(location, "_");
            return Math.max(location - 1, 0);
        } else {
            board.set(location, userInput);
            return location + 1;
        }
    }

    private static String getInput(String prompt) {
        System.out.print(prompt + " ");
        Scanner scanner = new Scanner(System.in);
        return scanner.nextLine();
    }

    private static void display(List<String> currentBoard) {
        System.out.println(currentBoard);
    }

    private static void resetBoard() {
        board.clear();
        for (int i = 0; i < 5; i++) {
            board.add("_");
        }
    }
```

```
    public static void play() {
        String prompt = "Your current cell is marked with a
        '='.\n * Enter a letter if you know it belongs in that
        cell.\n * Enter a space if you want to skip to the next
        cell.\n * Enter '1' to move one cell back and make
        changes.\n * Enter '0' to exit the board editing
        process.";
        System.out.println(prompt);

        resetBoard();
        initializeBoard();

        List<String> rows = letterWords();
        List<String> results = matcher(rows, board);
        String must = getInput("Type in the letters to include
        but not sure of location: ");
        results = mustHave(results, must);

        System.out.println("Here are the results of the
        matches:");
        System.out.println(results);
    }

    public static void main(String[] args) {
      play();
    }
}
```

Playing the Game

The final step of the project is to modify the `play` method. Up until now, we used this method to test our program in various states of completion. We are not adding new features, so the next step is to solidify the user experience and bring all of the existing methods together into a cohesive program.

On This Page...

1. **Continual Play**: Create an environment that does not end after one iteration.
2. **Letters to Ignore**: Capture and handle letters to ignore.
3. **Required Letters**: Capture and handle required letters.
4. **End or Restart Solver**: Allow user to end or restart the solver.

Continual Play

As always, prepare the board for the game. The Wordle solver should run as long as it take to guess the word. As such, use a while loop that uses `true` instead of a conditional. Inside the loop, create the variable `rows` which represents all of the words in the dictionary.

```
    private static void play() {
        resetBoard();
        initializeBoard();

        while (true) {
            List<String> rows = letterWords();

        }
    }
```

Letters to Ignore

Next, we want to prompt the user to provide any letters to be ignored. However, we are only going to call the `remover` method if entered something at the prompt. After removing any words containing letters we should ignore, we are going to match the list of possible words (`rows`) against the state of the board and save this list as `results`.

```
    private static void play() {
        resetBoard();
        initializeBoard();

        while (true) {
            List<String> rows = letterWords();

            String ignore = getInput("Type in the letters to
        ignore (if any): ");
            ignoreWords = new ArrayList<>
        (Arrays.asList(ignore.split(",")));

            if (!ignoreWords.isEmpty() &&
        !ignoreWords.get(0).equals("")) {
                rows = remover(rows, ignoreWords.get(0));
            }

            List<String> results = matcher(rows, board);

        }
    }
```

Required Letters

Again, we are going the user, but this time for any letters that required. Update `results` with the `mustHave` method, and then print out the list of possible words. To keep things a little more orderly, we are going to print out, at most, ten words from the list.

```
    private static void play() {
        resetBoard();
        initializeBoard();

        while (true) {
            List<String> rows = letterWords();

            String ignore = getInput("Type in the letters to
        ignore (if any): ");
            ignoreWords = new ArrayList<>
        (Arrays.asList(ignore.split(",")));

            if (!ignoreWords.isEmpty() &&
        !ignoreWords.get(0).equals("")) {
                rows = remover(rows, ignoreWords.get(0));
            }

            List<String> results = matcher(rows, board);

            String must = getInput("Type in the letters to
        include but not sure of location: ");
            results = mustHave(results, must);

            System.out.println("Here are the results of the
        matches:");
            if (results.size() <= 10) {
                System.out.println(results);
            } else {
                System.out.println(results.subList(0, 10));
            }

        }
    }
```

End or Restart Solver

Finally, we want to give the user the ability to continue, quit, or to restart the program for another round of Wordle. Prompt the user, giving them the choices of input. If they want to end the game, break out of the while loop. The program will stop. If they want to restart, reset and initialize the board. If they want to continue, call the `initializeBoard` method.

```
    private static void play() {
        resetBoard();
        initializeBoard();

        while (true) {
            List<String> rows = letterWords();

            String ignore = getInput("Type in the letters to
        ignore (if any): ");
            ignoreWords = new ArrayList<>
        (Arrays.asList(ignore.split(",")));

            if (!ignoreWords.isEmpty() &&
        !ignoreWords.get(0).equals("")) {
                rows = remover(rows, ignoreWords.get(0));
            }

            List<String> results = matcher(rows, board);

            String must = getInput("Type in the letters to
        include but not sure of location: ");
            results = mustHave(results, must);

            System.out.println("Here are the results of the
        matches:");
            if (results.size() <= 10) {
                System.out.println(results);
            } else {
                System.out.println(results.subList(0, 10));
            }

            String nextMove = getInput("Do you want to continue
        (c), reset (r), or exit (e)? ");

            if (nextMove.equals("e")) {
                break;
            } else if (nextMove.equals("r")) {
                resetBoard();
                initializeBoard();
            } else if (nextMove.equals("c")) {
                initializeBoard();
            }
        }
    }
```

You now have a working Wordle solver. Compile and run the program. Be sure to test out all of the features, especially the ability to continue and restart the solver.

▼ Code

Your code should look like this:

```
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.util.List;
```

```java
import java.util.Scanner;
import java.util.ArrayList;
import java.util.Arrays;

public class WordleSolver {
    private static List<String> ignoreWords = new ArrayList<>();
    private static List<String> board = new ArrayList<>();

    private static List<String> mustHave(List<String> bigList,
        String must) {
        List<String> newList = new ArrayList<>();
        for (String word : bigList) {
            boolean flag = true;
            for (char ch : must.toCharArray()) {
                if (word.indexOf(ch) == -1) {
                    flag = false;
                    break;
                }
            }
            if (flag) {
                newList.add(word);
            }
        }
        return newList;
    }

    private static List<String> remover(List<String> bigList,
        String ignore) {
        List<String> newList = new ArrayList<>();
        for (String word : bigList) {
            boolean flag = true;
            for (char ch : ignore.toCharArray()) {
                if (word.indexOf(ch) != -1) {
                    flag = false;
                    break;
                }
            }
            if (flag) {
                newList.add(word);
            }
        }
        return newList;
    }

    private static List<String> matcher(List<String> bigList,
        List<String> target) {
        List<String> results = new ArrayList<>();

        for (String word : bigList) {
            boolean isMatch = true;
            for (int i = 0; i < 5; i++) {
                if (!target.get(i).equals("_") &&
        !target.get(i).equals(String.valueOf(word.charAt(i)))) {
                    isMatch = false;
                    break;
                }
            }
            if (isMatch) {
```

```
                results.add(word);
            }
        }

        return results;
    }

    private static List<String> letterWords() {
        List<String> rows = new ArrayList<>();
        try (BufferedReader reader = new BufferedReader(new
        FileReader("wordle-solver/words.txt"))) {
            String line;
            while ((line = reader.readLine()) != null) {
                rows.add(line.trim());
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
        return rows;
    }

    private static void initializeBoard() {
        int i = 0;
        while(i < 5) {
            i = userEntry(i);
        }
        display(board);
    }

    private static int userEntry(int location) {
        String initialState = board.get(location);
        board.set(location, "=");
        String prompt = "Your current cell is marked with a
        '='.\n * Enter a letter if you know it belongs in that
        cell.\n * Enter a space if you want to skip to the next
        cell.\n * Enter '1' to move one cell back and make
        changes.\n * Enter '0' to exit the board editing
        process.";
        System.out.println(prompt);
        String userInput = getInput(board.toString());

        if (userInput.equals(" ") || userInput.isEmpty()) {
            board.set(location, initialState);
            return location + 1;
        } else if (userInput.equals("0")) {
            board.set(location, initialState);
            return 5;
        } else if (userInput.equals("1")) {
            board.set(location, "_");
            return Math.max(location - 1, 0);
        } else {
            board.set(location, userInput);
            return location + 1;
        }
    }

    private static String getInput(String prompt) {
        System.out.print(prompt + " ");
```

```
        Scanner scanner = new Scanner(System.in);
        return scanner.nextLine();
    }

    private static void display(List<String> currentBoard) {
        System.out.println(currentBoard);
    }

    private static void resetBoard() {
        board.clear();
        for (int i = 0; i < 5; i++) {
            board.add("_");
        }
    }

    private static void play() {
        resetBoard();
        initializeBoard();

        while (true) {
            List<String> rows = letterWords();

            String ignore = getInput("Type in the letters to
        ignore (if any): ");
            ignoreWords = new ArrayList<>
        (Arrays.asList(ignore.split(",")));

            if (!ignoreWords.isEmpty() &&
        !ignoreWords.get(0).equals("")) {
                rows = remover(rows, ignoreWords.get(0));
            }

            List<String> results = matcher(rows, board);

            String must = getInput("Type in the letters to
        include but not sure of location: ");
            results = mustHave(results, must);

            System.out.println("Here are the results of the
        matches:");
            if (results.size() <= 10) {
                System.out.println(results);
            } else {
                System.out.println(results.subList(0, 10));
            }

            String nextMove = getInput("Do you want to continue
        (c), reset (r), or exit (e)? ");

            if (nextMove.equals("e")) {
                break;
            } else if (nextMove.equals("r")) {
                resetBoard();
                initializeBoard();
            } else if (nextMove.equals("c")) {
                initializeBoard();
            }
        }
    }
```

```
    public static void main(String[] args) {
      play();
    }
}
```

Recap

Play Wordle

Now that we have a working Wordle solver, let's use it with the actual game. Click the button below to start the solver in the terminal (bottom left), and use it with the game (top right). Here is the feedback provided by Wordle:

- **Green:** The letter is in the correct position.
- **Yellow:** The letter is in the word, but not in the correct position.
- **Gray:** The letter is not in the word.

Well Done

Congratulations, you've made it to the end of this project! We've walked through the different aspects of building a Wordle solver in Java, dissecting each function and exploring its role in the broader context of the application. Essentially, creating your first coding project.

To review, here are the key components to our Wordle solver:

- **Initializing the board**: Via `resetBoard()` and `initializeBoard()`.
- **Fetching a list of possible words**: Through the `letterWords()` function.
- **Filtering out words based on conditions**: Using `matcher()`, `remover()`, and `mustHave()`.
- **User interaction**: Via the `getInput()` method.
- **Main gameplay**: Orchestrated by the `play()` function.

The code provided is designed to be flexible and extensible. You can easily extend or adapt it to fit different requirements. For example, you can:

- Implement a GUI
- Use different sorting and searching algorithms
- Further refine the word filtering algorithms
- Integrate it into a web application.

Pushing to GitHub

Pushing to GitHub

Now that we have finished the Wordle solver, we need to push it to GitHub so that you have a copy in your portfolio.

1. On the left side, you should see a file tree with the `wordle-solver` folder.
2. Click on your `code` folder and copy its contents into your git `wordle-solver` created folder.
3. In the terminal, run the command `cd wordle-solver` to move into the folder connected to GitHub.

- Now we can Commit your changes:

```
git add .
git commit -m "Finished the Wordle solver"
```

- Push to GitHub:

```
git push
```