…Be PROACTIVE and seek help EARLY in the week rather than panicking on the due date.

ALL work on this project is to be done individually. Seeking help is fine but working too closely with other students will result in a violation of the academic policy for all students involved (*Unauthorized Collaboration*). This obviously also includes referencing solutions from ANY external websites (*Unauthorized Aid*). If you use code YOU did not develop, *you must cite your source* in a comment above the line(s) of code indicating it is NOT your own work, otherwise it will be considered *plagiarism* (*representing others' work, whether copyrighted or not, as one's own*). *Violate KSU Academic Integrity policies at your own risks – sanctions are severe if caught.*

CIS 200: Project 10 (50 points + 20% Extra Credit)

Due: Friday, May 6th by 11:59pm

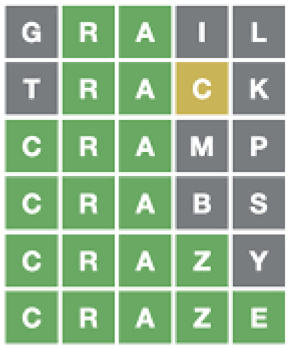
Since May 6th is the last official day of the semester, this project WILL NOT BE ACCEPTED LATE (to allow the GTAs time to grade before the final exam.)

Background: *Wordle* is a game developed by Josh Wardle, a software engineer at Reddit, who created it for his partner, who loves word games. It was publicly released in Oct 2021 and has become immensely popular.

The concept is similar to the *guess-the-color* game *Mastermind*. Wordle is a grid of five squares across and 3-6 rows down, depending on the difficulty level. Each horizontal row of five squares represents an opportunity to guess the random, five-letter word of the day. With six rows, players have six opportunities to guess.

With each guess, the game will let you know which letters you have right. The background of a letter turns green if it is in the word and in the correct spot, or yellow if it's contained in the word, but in the wrong spot. Those that are not contained in the word are dark gray.

Each guess, and each correct letter, gets you closer to finding the word of the day. And it can get tricky — sometimes letters are used more than once. The goal is to guess the word in as few tries as possible and before you run out of attempts.



For more information, just google *Wordle*. Here is a very short *YouTube* video on *How to Play Wordle*
https://www.youtube.com/watch?v=lv4Zg-209MY&ab_channel=TripleSGames

---

Assignment Description: Create a *non-GUI (Console) C# application*

To demonstrate the similarities between *Java* and *C#*, you will create a simplified, console I/O version of *Wordle*. It will consist of at least two .cs files – *Words.cs* and *Wordle.cs*. You may have more, but this is the minimum.

The design of the application will be left to each of you. The only requirement is the *Words* class must be used to read in all the 5-letter words in the provided *words.txt* file and store the words in an array or array list within the class. Manipulate of this list of words should be done in the *Words* class, but that is up to you. The *Wordle* class is the application class that will be the executable file the user clicks on to play the game.

This project can be done using a text editor with the C# command line compiler ***or*** using *Visual Studio*. Regardless of which method you use, submit ONLY the two (or more) .cs files within a zipped ***Proj10*** folder.

Note: If doing the MVC extra credit, please read the requirements for the extra credit below first, since program development would be different for the *extra credit* versus the required project.

Implementation Requirements:

Within the *Words* class: (You may do more in the *Words* class, as you deem necessary. Remember that manipulation of the list of words or any single word within the list would best be done within the *Words* class.)

- From the given *words.txt* file consisting of unique, uppercase, 5 letter words, load all the words contained in the file into an array or array list.
- Randomly pick one of the words in the list by generating a random number between 1 and the number of words contained in the array/array list. Return the word that matches that *index-1*. For example, if 98 is randomly generated, you would return the 97th word in the list.

Generating Random Numbers in C#:

```
Random rnd = new Random();
int month   = rnd.Next (1, 13);   // creates a number between 1 and 12
int dice    = rnd.Next (1, 7);    // creates a number between 1 and 6
int card    = rnd.Next (52);      // creates a number between 0 and 51
```

Within the *Wordle* class

- Allow the user to enter a Skill Level that will correspond to how many rows are provided in their *wordle*: *Beginner*-6 Rows, *Proficient*-4 Rows, or *Expert*-3 Rows.
- Ignore the case of the word, so Hello = HELLO = hello. I would suggest handling ONLY uppercase / all caps. Word chosen and all letters entered by the user should be converted to uppercase.

IMPORTANT: For testing purposes (for both you and the GTA), display "*Chosen Word:* " followed by the word chosen by the random word generator. Leave this test UNCOMMENTED so GTA can properly test your program.

- User will enter a 5-letter word. (Although it is supposed to be a *real* word, for our simplified application, allow *any* 5 letters.)
- Once the user enters the word, perform the following check and display accordingly:

  -If a letter is in the correct position in the word the user is trying to guess, display that letter in that position.

  -If a letter is in the word but not in the correct position, display a * in the position the user guessed.

  -If a letter is NOT in the word, display a '-' (hyphen) followed by a space in the position user guessed.

For example, let's say the random word chosen for a *beginner* is HOARD.
(A beginner would have 6 guesses before the word is displayed and the game ends)
Input shown in red for clarity…yours will NOT be in red

Enter guess 1: CLEAR …A & R are in the word, but not in the right position
- - - * *

Enter guess 2: ADORE …ADO are in the word, but not in the right position … R is in the right position
*****R-**

Enter guess 3: **ROADS** …RD are in the word, but not in the right position … OA are in the right position
***OA*-**

Enter guess 4: **HOARD** …Word matches so we have a winner!

- After the user plays the game, display if they WON or LOST, then ask if they wish to play again. Allow them to re-enter the skill level if playing again. Play until the user chooses not to play.

Documentation: Put a description at the top of each class file. Since this is the last project, comments at the top of each method are *optional* (but always a good idea).

```
/**
* <Full Filename>
* <Student Name / Lab Section Day and Time>
*
* Clearly indicate to the reader of your code what THIS class does (i.e., the purpose of THIS class)
        Detailed enough so outside reader of code can determine the purpose of this class (at least 3-4 sentences)
*/
```

Running Requirements:

This project will contain at least TWO classes – *Words.cs* and *Wordle.cs*.

All classes must compile (by command-line) with the statement: **csc filename.cs (or csc *.cs)**

It must then run with the command: **Wordle**

****Make sure and test this before submitting, so points are not lost unnecessarily.**

OPTIONAL Extra Credit Challenge: (20% extra credit)

If you do either of the extra credit, please indicate in your initial comments on p.1 of your *Wordle* file "OPTIONAL EXTRA CREDIT #1 (and/or #2) INCLUDED" (so the TA is aware before they grade it).

Do *either* or *both* of the extra credit listed below:

1) (+4%) Add the following modification to Project 10. Submit a SINGLE version of this Project to Canvas. Follow the rules of *MVC architecture* by creating a separate class to handle the VIEW (**Console I/O**). Then use the VIEW class to handle ALL I/O within the Project – *no WriteLine statements or input statements in the Controller class* (*Wordle*) or Model Class (*Words*). All communication between the *model* and *view* classes is done through the *controller* class. Submit all THREE .cs files in a zipped Proj10 folder.

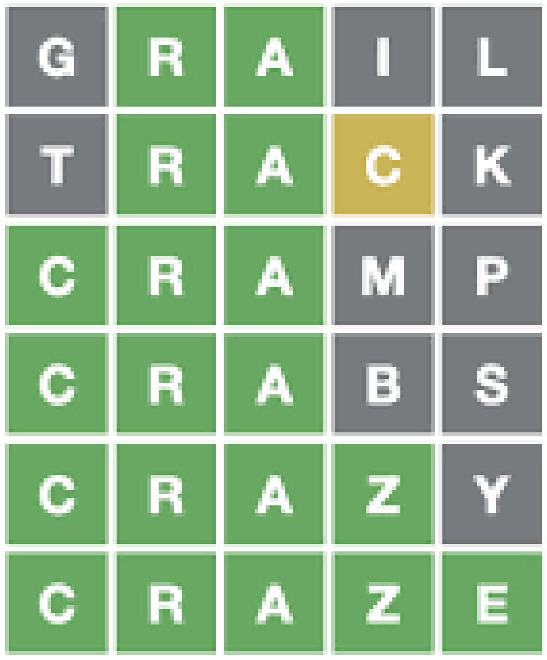
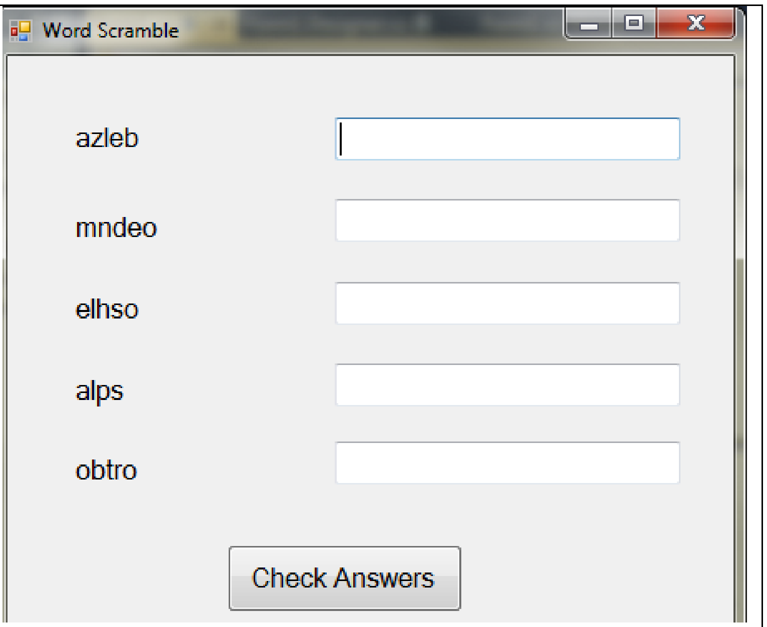
2) (+16%) Using *Visual Studio*, create a separate GUI interface to play Wordle. The GUI interface does NOT have to look like the GUI interface shown on p.1, as long as it works as instructed.

   FYI: A C# GUI already uses MVC architecture–*Words model*; *Form1.Designer View*; *Form1 controller*.

Suggestions on the C# GUI Extra Credit:

- Further modification of your *Words* (.cs) class should not be necessary. Simply add it to your Visual Studio project (refer to the *CompileRunGuide_C#.pdf* posted within Canvas under Files-"C# Reference")
- You are, basically, on your own in designing/implementing the optional extra credit GUI. Your best source of information would be the internet. The current lab UTAs will be the most familiar with C# GUI, since they work with it extensively in CIS 300, so they are another source of help.
- All of your modifications will be done with the file *Form1.cs*. There is a sample *Form1.cs* file available in Canvas (within the assignment posting) called "*Word Scramble_Form1.cs*" used to create the GUI on the next page. There is also a *Tic-Tac-Toe* example posted in Canvas in which only a single handler/method is used.
- When creating the GUI interface within Visual Studio, a **.cs** file named *Form1.Designer* will be automatically populated with all of your GUI information. You will NOT need to modify this file manually.

<u>Possible GUI Designs</u>

 Your GUI interface does <u>NOT</u> have to look/work exactly like the above.	 Word Scramble azleb mndeo elhso alps obtro Check Answers Just an idea…GUI used on a previous project. Use of textboxes to get user input, and a button used as *enter*. Could have another textbox to display the current results. (See *Word Scramble_Form1.cs)*

If doing the extra credit GUI, remember <u>the required portion that plays the game using CONSOLE I/O must also be submitted</u>. Test your program on a WINDOWS machine before submitting. GTAs will be testing this project and GUI Extra Credit in a *Windows* environment, so it is the student's responsibility to make sure it runs in a *Windows* environment. This is especially important for MAC users.

Submission – <u>read these instructions carefully or you may lose points</u>

To submit your project, <u>first create a folder called *Proj10*</u> and copy ALL *.cs* files (<u>only the .cs files</u>) into that folder. Then, right-click on that folder and select "*Send To* → *Compressed (zipped) folder*". This will create the file *Proj10.zip*. Only a .zip file will be accepted for this assignment in Canvas.

If doing the extra credit GUI, zip the ENTIRE project folder created by Visual Studio <u>and submit separately</u>. This is in addition to the files requested for the Console Version of the application. Make sure the **.exe** file is contained in the *\bin\debug* folder of your Visual Studio project. GTAs will be clicking on this file to test your application. They will NOT be getting into Visual Studio to test.

Important: It is the *student's responsibility* to verify that the <u>*correct*</u> file is <u>*properly*</u> submitted. If you don't properly submit the *correct* file, *it will not be accepted after the due date passes*. <u>No exceptions</u>.

Grading: You must submit *all required classes (at least 2)*, and ALL must compile at the command line to be considered for partial credit (i.e., submitting only the *Words* or *Wordle* class would not earn you any points).

If doing the GUI extra credit, remember to submit the ENTIRE Visual Studio Project separately as a .zip file. Make sure that the *exe* file is included and what you submit is runnable by the GTA.

Programs that do not compile will receive a grade of 0. Programs that *do* compile will be graded according to the following rubric:

Requirement	**Points**
Points awarded only on C# submissions – minimum of Words.cs and Wordle.cs	**-**
Documentation – includes correct documentation on EACH file (class)	**3**
Words Code: Min includes an Array or ArrayList as a data property with the contents of words.txt loaded into the Array/ArrayList and a method that returns a random word from the Array/ArrayList	**7**
Execution (40 pts.)	**-**
Allows user to enter a Skill Level	**1**
"Chosen Word" to be guessed is displayed for testing purposes	**2**
Game works as instructed, depending on if the letter is in the correct position (display letter), not in the correct position but is in the word (display *), or not in the word (hyphen-space); correctly determines if user wins or loses.	**20**
Allows the correct number of correct number of rows based on the chosen Skill Level (6, 4, or 3)	**3**
Repeats the game as many times as user wishes to play	**2**
If user plays again, different Skill Level can be entered. Game works as instructed on repeated plays.	**12**
Extra Credit Option: *(Converted to MVC)* (+4%) Includes an additional View class added to do ALL I/O / *Words* and *Wordle* classes includes NO direct *input* or *output* statements	**+2**
Extra Credit Option 2: *(Visual Studio Projects only)* (+16%) Wordle GUI interface created and works as it should	**+8**
****NOT ACCEPTED LATE**	**-**
Total	**50 + 10**