

The Computational World of Wordle

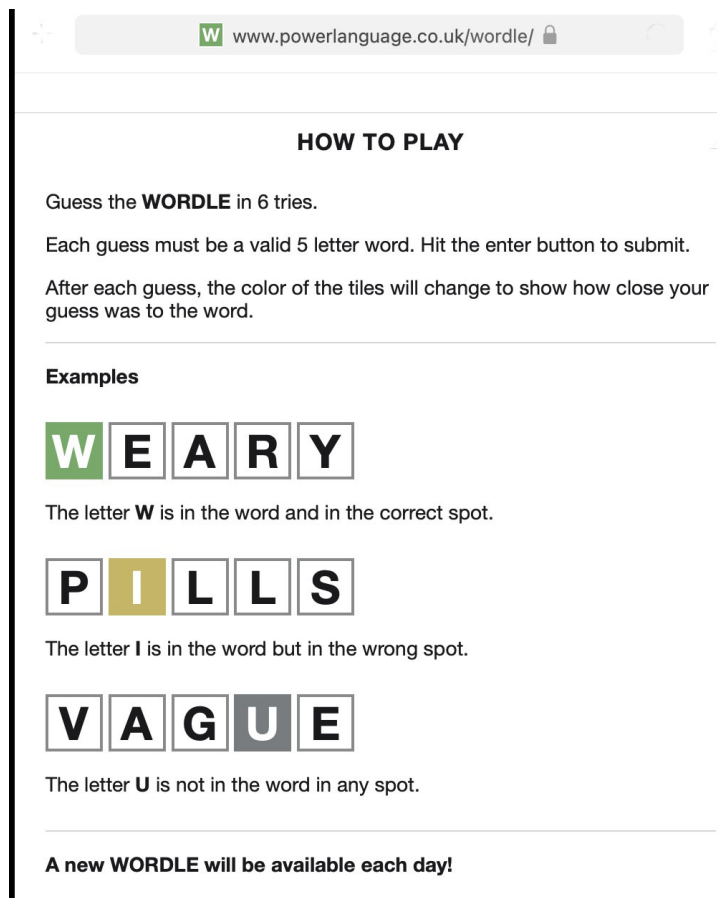
publication date XXXXX

David Reiss, Senior Principal Consultant, Wolfram Solutions

What is Wordle? The online game Wordle has come at a time when each of us needs a simple, friendly challenge to take our minds off of other worldly issues.

Josh Wardle is the person who created Wordle for his partner and then shared it with the world . You can play it [here](#).

The concept is simple and engaging. You are challenged to guess a five letter word in six guesses. Here are the rules from the Wordle website:



The screenshot shows a web browser window with the URL www.powerlanguage.co.uk/wordle/. The page title is "HOW TO PLAY". The instructions state: "Guess the **WORDLE** in 6 tries. Each guess must be a valid 5 letter word. Hit the enter button to submit. After each guess, the color of the tiles will change to show how close your guess was to the word." Below this, there are three examples of guesses and their feedback: 1. Guess: WEARY. Feedback: The letter W is in the word and in the correct spot. (W is green, others are grey). 2. Guess: PILLS. Feedback: The letter I is in the word but in the wrong spot. (I is yellow, others are grey). 3. Guess: VAGUE. Feedback: The letter U is not in the word in any spot. (U is grey, others are grey). At the bottom, it says "A new WORDLE will be available each day!"

That there is only one new Wordle a day is a sensible design choice—it gives you a new game once a day and protects you from any tendencies you might have to get stuck playing it over, and over, and over...

That's the idea at least.

The challenge of a slow weekend I was texting with my daughter (she had introduced me to Wordle a few days earlier):

I think that I should write a version of wordle in Mathematica

I'm sure I can do it

That would be so cool!

My colleagues and I at [Wolfram Solutions](#) build some pretty large and complex user interfaces for our customers (a recent one comes to about 25,000 lines of [Wolfram Language](#) code just for the user interface portion alone). So I am pretty confident when it comes to putting together a UI quickly. I decided to take up the challenge in order to keep myself busy during that slow weekend.

I few hours later I texted my daughter back with an initial version:

Sun, Jan 9, 9:24 PM

I wrote an initial version of something like world 🤪 Sleep tight

tight

Mon, Jan 10, 9:38 AM

❤️ i was asleep, so sorry! but yay!! (and i didn't want to text you too early)

No worries, that's what I figured 😊 I can show you tonight

New

T	E	S	T	S
S	A	X	E	S

Next

Yes!, The word is: SAXES

Show answer

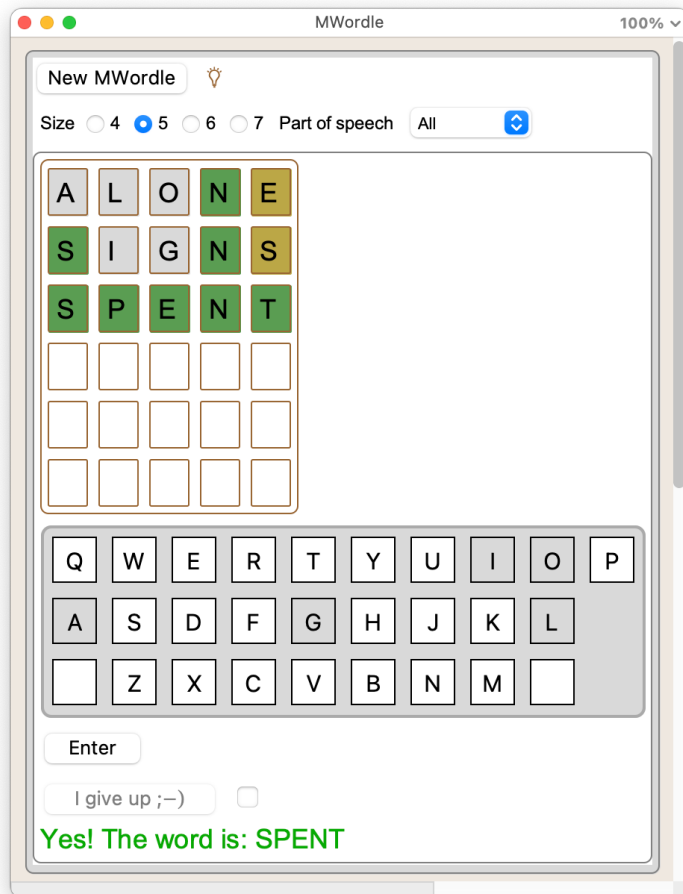
"SAXES"

As you can see, I am a bit spelling-challenged (this figures later in the story, along with the fact that my daughter is a speech pathologist...).

Now, in creating this, right away I violated the sacrosanct principle of “only one Wordle A day”. All I can say is that I hope anyone who gets hooked on this will forgive me.

I [posted a message](#) on this to the [Wolfram Community](#) to share the code with others, and also so that they could play with the code as well as the application itself. (You can read the full code and download the package from that post, as well as see some other folks' comments.) It's also an example of one way to design and code a Wolfram Language [package](#).

Over the next week I spent a little time tweaking the application to let the user choose what part of speech the word is restricted to, as well as to give them the choice of whether the length of the word is 4, 5, 6, or 7 characters long.



(Now, if you go to the Wolfram Community post you will see an animated GIF of the application. In its original version I misspelled “speech” as “speach”: given that my daughter is a speech pathologist, I should have known better! And it’s not her fault that she didn’t tell me of the error because I hadn’t yet given her the updated version.)

These additional bits of functionality, as compared to the original Wordle, are out of line with the (brilliant) simplicity of its design. But it makes the point that the both the algorithmic and user interface capabilities of the Wolfram Language let you explore variations and approaches to your heart’s content.

The MWordle.m package comes to a bit more than 400 lines of Wolfram Language code. The JavaScript code of the web version is quite a bit more than this (but, to be fair, it has

more functionality than the Wolfram Language facsimile that I wrote). Using the Wolfram Language's vast resources you can customize, revise, and debug versions ad infinitum, and very efficiently. Though it was a slow weekend, after I created the first version of this, I still had plenty of time to take care of doing my laundry, catching up on Netflix, and then coming back to tweak the code.

And more... Aside from creating the facsimile, interesting questions come up about strategies for playing the Wordle game. As expected, there is much chatter on the internet about this. What are the best words to use when making your first guess? How can you optimize the subsequent guesses? And so on, and so on...

Arnoud Buzing quickly created a [ResourceObject](#) that contains the actual word list that the web version of Wordle uses. (My code uses Mathematica's dictionary through the [WordData](#) function.)

While I personally prefer to leave a patina of mystery to playing the game, and approach the online version as if I was an algorithmically naive person, it's incredibly straightforward to explore things with the Wolfram Language. Like many other things, a simple game like Wordle can be viewed as starting point for exploring a particular computational world.

Here are some examples. Someone, in my Wolfram Community post asked "I was thinking it would be good to compute what the best starting word would be for Wordle based on [WordData](#), letter frequencies and letter position frequencies".

So I took up the challenge and wrote the following as an example of one possible starting point—initially without taking into account the positions of the letters.

Here are all the 5 letter words that are used in the application:

```
In[ ]:= $fiveLetterWords =
Module[{words}, words = Select[
  WordList["KnownWords", IncludeInflections → True], StringLength[##] === 5 &];
words = ToUpperCase /@ words;
words = Select[words, StringMatchQ[##, LetterCharacter ..] &];
Union[words]
];
```

There are 7517 of them.

```
In[ ]:= Length[$fiveLetterWords]
```

```
Out[ ]:= 7517
```

Here is the ordering of the frequency of English letters for these 5 letter words:

```
In[ ]:= $orderedLetters = Keys[ReverseSort[Counts[Flatten[Characters /@ $fiveLetterWords]]]]
```

```
Out[ ]:= {S, A, E, O, R, I, L, T, N, U, D, C, P, M, H, Y, B, G, K, F, W, V, X, Z, J, Q}
```

So let's see if there are any words amongst the list of these 5 letter words that match the highest 5 of the frequency sorted letters (and choosing that there be no repeated letters

in the word):

```
In[ ]:= StringJoin /@ Select[Characters /@ $fiveLetterWords,
      Sort[Intersection[#, $orderedLetters[;; 5]]] === Sort[$orderedLetters[;; 5]] &
    ]
```

```
Out[ ]:= {AROSE}
```

Wow, there's only one! And it takes care of another possible approach—try to have as many vowels as possible.

So, let's relax the constraint slightly and pull things from the highest **nLetters** letters in the frequency ordered list, but still make sure that there are no repeating letters.

```
In[ ]:= findWordlePossibilities[nLetters_Integer] :=
  Module[{possibilities},
    possibilities =
      Select[Characters /@ $fiveLetterWords,
        SubsetQ[$orderedLetters[;; nLetters], #] &];
    StringJoin /@ Select[possibilities, Length[Union[#]] === 5 &]
  ]
```

From the top 5 letters, as before:

```
In[ ]:= findWordlePossibilities[5]
```

```
Out[ ]:= {AROSE}
```

From the top 6 letters:

```
In[ ]:= findWordlePossibilities[6]
```

```
Out[ ]:= {AESIR, ARIES, ARISE, AROSE, OSIER, RAISE}
```

From the top 7 letters:

```
In[ ]:= findWordlePossibilities[7]
```

```
Out[ ]:= {AESIR, AISLE, ALOES, ARIES, ARILS, ARISE, ARLES, AROSE, EARLS, ELISA, LAIRS, LASER,
  LIARS, LOIRE, LOIRS, LORES, LOSER, OILER, ORALS, ORIEL, OSIER, RAILS, RAISE,
  RALES, RIALS, RIELS, RILES, ROILS, ROLES, SELAR, SLIER, SOLAR, SOLEA, SOLER}
```

Arnoud also [posted a blog](#) on an approach to optimizing the guesses that you might make as you work through a Wordle challenge. In it he takes into account the letter frequencies based on the positions of letters in the word.

And [Peter Barendse](#) suggested that code like the sort that I have in the MWordle application could be used to train an intelligent agent to play Wordle. That'd be fun to do.

The possibilities are endless and the Wolfram Language is the ideal vehicle for exploring the [Computational World](#) of Wordle.