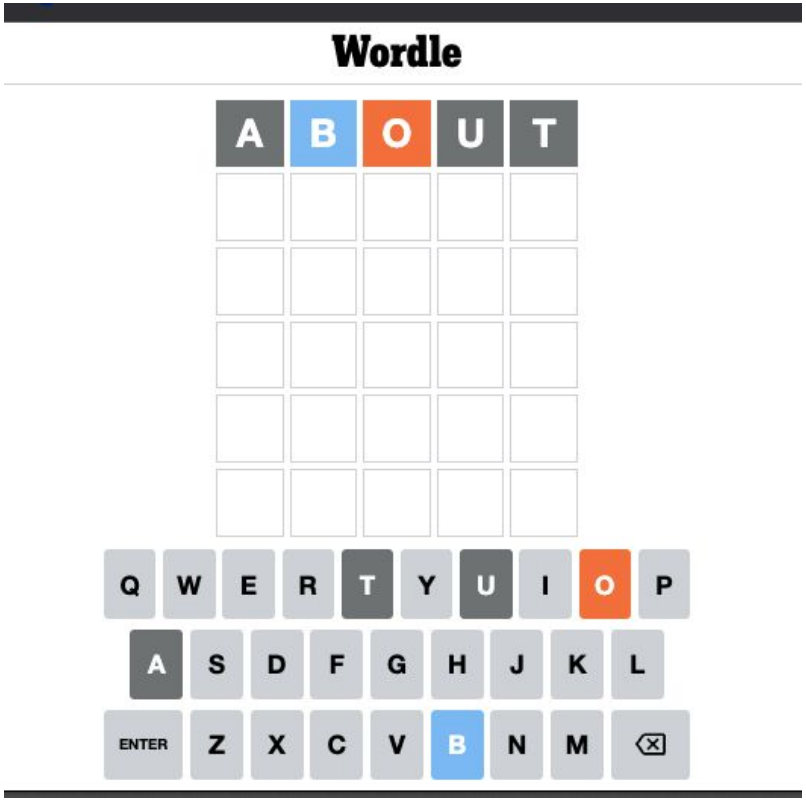
Wordle better with code (and a little bit of math)

Rama Kocherlakota
Cox Automotive
Level Up 2022
2022-09-27

Wordle

- You probably know what Wordle is - it's puzzle game that took the internet by storm beginning in late 2021.
  - I've heard it described a the sourdough starter of omicron
- The basic idea is that there is a five letter word that you have to guess - you have six guesses and after each guess you get back feedback about how close you are to guessing it correctly.

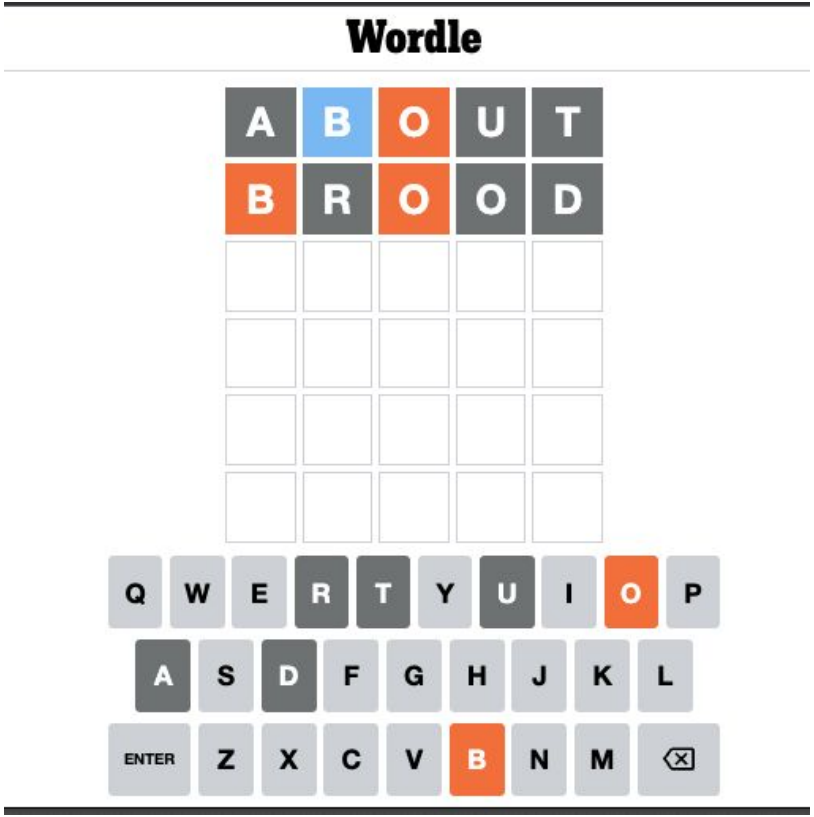
Here's an example:



- Say you guess "ABOUT" and the program responds by coloring the guess to show which letters are right and which are almost right.
- Orange means "yup, right letter, right location", light blue means "that letter is right but it's not in the wrong spot
- So this word has an O in the middle and a B somewhere in it but not in the second spot
- also, and maybe most importantly, we know that A, U and T are not in the answer anywhere

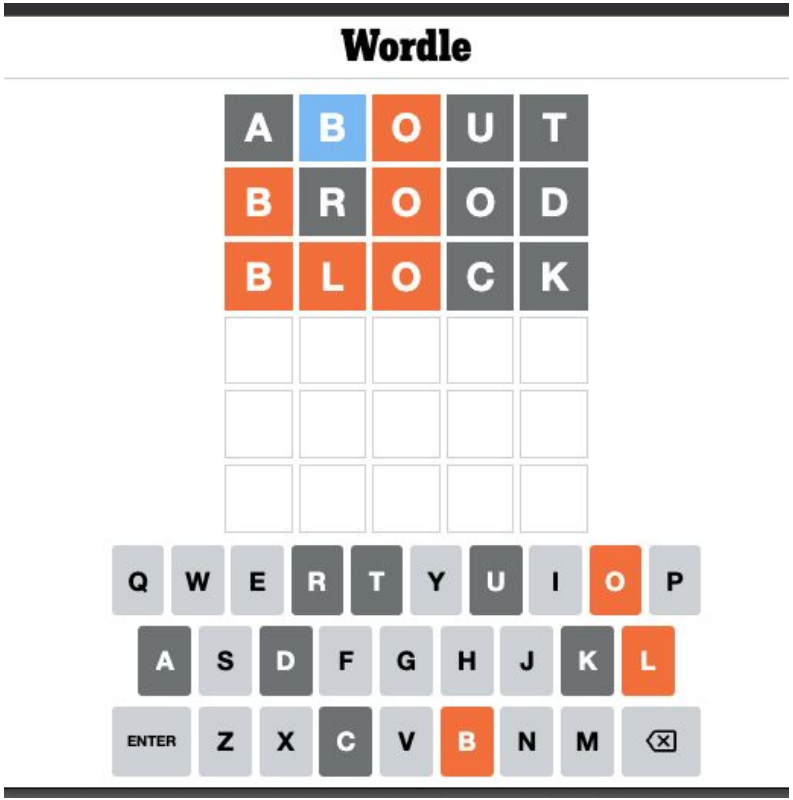
(I play Wordle in High-Contrast mode, so my colors might be different from what you're used to but that won't make any difference for most of what follows.)

Second guess…



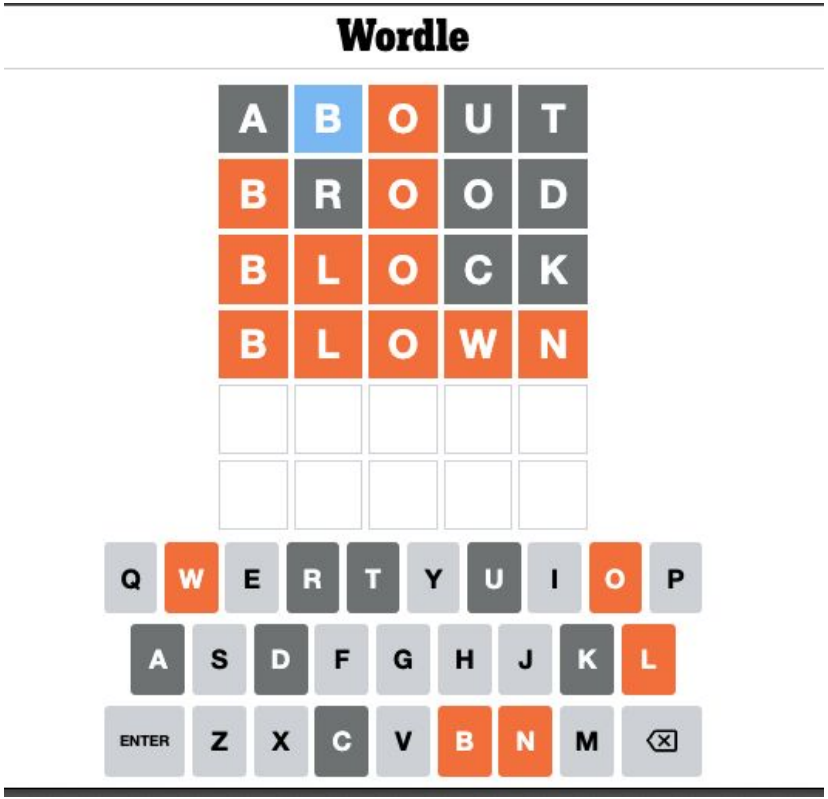
- we want a word that looks like "--O–" with a B in it somewhere. How about "BROOD?"
- Okay, getting warmer. Now the program says we have the B and the O in the right place, plus there's no R or D and no second O.
- Let's try… BLOCK

Third guess, close but no cigar



- Oh, boy, almost there!
- BLO–... what could it be? And no C or K, so it's not BLOKE
- I feel like we're super close
- I bet it's BLOWN

Fourth time's the charm!



- yup, that's it!
- not bad, getting it in four, that's about my average, actually. Three on a good day, four on an average day. Five or six if it's a hard word or I'm struggling.
- I think that's true for a lot of Wordlers

Guessing games

- Wordle is a guessing game - there's an answer you need to figure out, you make a series of guesses and you get back clues as to how close you are to the answer
- It's a souped-up version of a game you probably played on long car trips as a kid. Somebody thinks of a number between 1 and 10 and you have to guess it. After each guess, they tell you "higher" or "lower"
- it's not much of a game as games go but there's some strategy: once you've played it three or four times you probably figured out that at each step of the way you should guess the midpoint of the remaining possibilities

Why bisect?

- Why is that? Why guess the midpoint? It's clearly the right thing to do, but why?
- Figuring this out will take us into a branch of math called "information theory" and help us to understand other guessing games, like Wordle, and come up with some strategies to help with that
- plus you'll get some bragging rights to your older sibling who probably needled you on those long car trips about how much better they were at the guessing game. And yes, I am a younger sibling, how did you know?

Information Theory

- Information theory was created by engineers and mathematicians at Bell Labs in the 1940's and 1950's, most notably Claude Shannon, who really got things rolling in 1958 in a book called the Mathematical Theory of Communication
- The most important notion in Information Theory is *entropy* or *uncertainty*. Information is interpreted as a reduction in uncertainty.
- So what is this uncertainty?

Uncertainty

- Let's start with the most classical guessing game of them all: the coin flip. I flip a coin, you call it in the air. Assuming the coin is fair, you have a fifty-fifty shot of guessing ir right. Let's call the amount of uncertainty you're dealing with *one bit*.
- Now suppose I flip two coins at once and you call them both. If the coins are fair and the flips are independent, that should have twice as much uncertainty (we haven't actually said what uncertainty is yet but bear with me).
- And for three coins? Three times as much uncertainty, surely.
- More generally, the uncertainty of two (or more) independent events should be the sum of the each of the uncertainties

Computing Uncertainty : Simple Cases

- Now, I'm going to give you a formula for uncertainty that obeys this intuitive additive property. Let's start with the simplest case, an experiment X with N possible outcomes, all of them equally likely. Take the definition of uncertainty to be H(X) = log(n)
- (In information theory, you always take your logs ot the base 2. It doesn't actually make much difference, it's basically just a choice of units)
- So, for the single coin flip, there are two equally likely outcomes, so H = log(2) = 1 (bit)
- Two coins, four outcomes, H = log(4) = 2 bits
- Three coins, H= log(8) = 3 bits

The number guessing game

- Let's look at the number guessing game. At the beginning, before you have any clues, here are ten equally likely possibilities, so the uncertainty is log(10) = 3.32 bits (which makes sense, it's a little more than the uncertainty in the three coin toss where there are eight possible outcomes)
- So now suppose you make a guess, say you guess n. Your older sibling in the seat next to you snickers and says "lower." What is the uncertainty now?
- You know the number must be one of 1, 2, … n-1 so the uncertainty is log(n-1)
- Conversely, if they say "higher" then the uncertainty is log(10-n) because you know the number must be one of n+1, n+2, … 10

So what's the expected uncertainty?

- If you guess n, the probability that they say "higher" is (10-n) / 10, and the probability that they say "lower" is (n-1)/10 (which adds up to 9/10 because there's always the 1/10 chance that you actually got it right but how likely is you older sibling to admit that?)
- So if you guess n, the expected uncertainty is

$$E(H(n)) = (n-1) \log (n-1) / 10 + (10 - n) \log (10 - n) / 10$$

Calculating…

- so, guessing around the middle, 5 or 6, is he right thing to do if you want to minimize the expected uncertainty after your guess

n	E(H(n)
1	2.853
2	2.400
3	2.165
4	2.026
5	1.961
6	1.961
7	2.026
8	2.165
9	2.400
10	2.853

Minimizing expected uncertainty

- This is the important point and one that I'm going to keep coming back to. We want to choose our guesses so as to minimize the expected uncertainty after we get back our response.
- Let's think about this carefully. You're about to make a guess and you know there are some number of possible right answers, so you need to choose one. There's some likelihood that you'll guess right but that's not the interesting situation for us right now - we're interested in what happens if you guess wrong
- We want to look at all the responses that could come back and optimize our guess so that we get as much information as possible from the response. The important point is that even if the answers are equally likely to be correct (which we've assumed) the responses aren't equally likely to come back
- By making the right guess, we can shade the distribution of responses to optimize the expected uncertainty after we get that response

Questions?

Wordle and Information Theory

- before I dig into how all this is related to Wordle, here's a plug for a video that explains it all really well. They have a different viewpoint from mine and the question they are trying to answer is a little different but it's worth watching: https://www.youtube.com/watch?v=v68zYyaEmEA
- So… Wordle has a set of possible answers and it lets you guess which one is the right one and gives you some information back after each guess
- Can we use what we learned studying the simple 1-10 guessing game to help us make better guesses?

How Wordle works

- there is a list of 2315 words that can be the answer to a Wordle puzzle
- there's a larger list of around 13000 words that are allowed guesses but I'm going to ignore that for the most part and just assume that guesses are to be drawn from the same list as the answers - it doesn't make much difference to the analysis and makes the computations go faster (plus many of the words on the larger list are words that only a Scrabble player could love. Nothing against Scrabble. I like Scrabble. But seriously… ADAWS?)
- Let's assume that on any given day, the Wordle word is selected randomly and uniformly from those 2315. That's not quite true - the words are fixed in advance and don't (so far) repeat, so if you remember yesterday's word it won't show up again today. Let's ignore that.

Wordle responses

- the way Wordle responds, by coloring letters, makes for a snazzy UI but isn't easy to encode. Instead, let's just treat a response as a sequence of `B` `W` and -, where `B` means "right letter, right place", `W` means "right letter, wrong place" and `-` means "not in the answer".

- this response would get encoded as "`-WB--`"
- (this coding system comes from the classic game Mastermind where "right but in the wrong place" was coded as a White peg and "right in the right place" was coded as a Black)

Wordle Uncertainty

- since there are 2315 different solutions to a Wordle puzzle, which means that that, under our assumptions of all of them being equally likely, the uncertainty before we've done any guessing at all is log(2315) = 11.177 bits.
- If all we were getting back as a response was a simple two-value thing like "higher" or "lower" then we would expect it to take something like eleven turns to get an answer, on average. But we only have six guesses, so it's good that Wordle gives us more information than that.

Putting the data where we can use it

- to facilitate doing some computations, we can put the guesses, answers and scores into a database table - that way we don't have to keep recomputing the score for a particular guess and answer combination.

```
select * from scores;
+--------+-------+-------+
| answer | guess | score |
+--------+-------+-------+
| merit  | micro | BW-W- |
| elfin  | lobby | W---- |
| mocha  | agree | W---- |
| guest  | repay | -W--- |
```

After one guess

- to figure out how many answers are compatible with the response we got from our ABOUT guess is a simple query

```
select count(*) from scores where guess='ABOUT' and score='-WB--';
+----------+
| count(*) |
+----------+
|       18 |
+----------+
1 row in set (0.01 sec)
```

- so our guess and the response have shrunk the set of possible answers from 2315 possibilities down to 18. This corresponds to a reduction in the uncertainty to log(18) = 4.170 bits

Is that good?

- so this guess (ABOUT) and this response (-BW- -) reduced the uncertainty from 11.177 to **4**.170 bits
- is that a good outcome? Specifically, is this about what we would expect over all responses if we guessed ABOUT?
- more generally, is ABOUT even a good first guess?

Remember the 1-10 guessing game…

- we found that the best guesses (5 or 6) were good because they optimized the *expected uncertainty after the response given the guess.*
- remember what this means: given our guess, we need to look at all of the possible responses, and compute
  - how much uncertainty we would have about the answer given the response
  - how likely the response is to come back

about ABOUT

- so, if we guess ABOUT, we've just seen that a response like `'-WB--'` corresponds to eighteen different answers. This means that if we get that response, there are log(18) bits of uncertainty about the answer and that the response has a probability of 18 / 2315 of coming back to us
- (that last part is because there are 2315 possible answers, and 18 of them would yield that response)
- all we need to do is take the sum over all the responses of the uncertainty given the response, weighting by the probability of the response

say it with SQL

- that's something we can get out of our `scores` table

```
select sum(c * log(2, c)) / sum(c) as h
  from (select score, count(*) as c from scores
    where guess='ABOUT' group by 1) as t1;
```

- the inner select is tabulating, for each response to ABOUT how many answers would give that response
- the outer select is then just taking the weighted sum of the log of that count, weighting by the probability of getting it back
- with the right indexes on the table, it returns super fast with an answer of 6.492 bits

more uncertainty calculations

- now remember that when we actually guessed ABOUT, Wordle gave us back a response of `'-WB--'` and we calculated that the uncertainty at that point was log(18) = 4.17 bits. Since the expected uncertainty is 6.492 bits, our guess did better than we would have expected, so we were lucky
- this leads us naturally to the question that people often ask after playing Wordle for a while

what is the best first guess?

- well, we can answer it. At least, using this expected uncertainty way of looking at the problem, the best guess is the one that minimizes the expected uncertainty. This lines up nicely with our guess-the-middle approach for the 1-10 game.
- the query for this is a variation on the one we just saw. To get the top 5:

```
select guess, sum(c * log(2, c)) / sum(c) as h from
  (select guess, score, count(*) as c
    from scores group by 1, 2 ) as t1
group by 1 order by 2 limit 5;
```

And the winner is…

- RAISE is the best first guess with an expected uncertainty of 5.3 bits

guess	h
raise	5.298886787326121
slate	5.321021101191640
crate	5.341922473884084
irate	5.345399497706812
trace	5.346247764288161

How good is ABOUT?

- remember that ABOUT came in with 6.49 bits of expected uncertainty. That, it turns out is just okay - out of the 2315 possible guesses we are considering, it is 1004th.
- there are plenty or worse first guesses - FUZZY, JAZZY, and MAMMA all come in at the end with about 8.9 bits of expected uncertainty.
- it's important to remember that this is an *expected* uncertainty value - when the right answer was BLOWN, remember that ABOUT did very well - the uncertainty after that guess was just 4.17 bits, whereas if you had guessed RAISE, the response would have been `'-----'` and the uncertainty would have been (it turns out) 7.39 bits. So even with the best first guess, you're still at the mercy of Lady Luck

what about SOARE? and ROATE?

- if you look on the internet, you'll find people touting SOARE and ROATE as the best starting words. And there's no question, they're good - in fact they're better than RAISE.
- if we expand our scores table to include all allowable guesses, not just the ones that are actually possible answers, we can get a list of the best guesses in that expanded set. And yes, SOARE and ROATE are at the top for expected uncertainty out of those But RAISE is next in line, and only by a hair (about a tenth of 1%)

```
| soare | 5.290836367768747 |
| roate | 5.294017153855633 |
| raise | 5.298886787326121 |
```

what about some other favorite first guesses?

- I googled "top wordle first guesses" and pulled some words out of the results. Some were pretty good, 'CRANE' almost made the top ten on my list. Others, less so. OUIJA is solidly down with JAZZY and MAMMA

guess	h	rank
crane	5.434014733450421	11
roast	5.5316030280865345	21
canoe	5.660140126098789	54
slice	5.662704400811293	57
tried	5.698665866046483	69
reach	5.8831422538443645	179
adieu	6.297969079052519	675
audio	6.587154941680242	1171
bakes	6.59119431726944	1174
ouija	7.265410645305752	1983

What about the second guess?

- even if we guess RAISE or something similar for the first guess, we aren't out of the woods - what do we do next? We need to keep guessing and we need to know what a good second guess is, ideally taking the response from the first guess into account. We have some choices:
  - using the response from the first guess determine the set of remaining possible answers and use the same kind of SQL query to figure out a guess that minimizes the uncertainty.
  - have a good second guess in mind from the start

strategy one: minimize expected uncertainty

- this is usually the best strategy, but it's hard to implement mentally. For instance, if the response from RAISE is `'-B---'` the optimal next guess is COUNT. Which is plausible, eliminates some other vowels, brings in the T and the N… but hard to know in advance.
- If instead the response is `'-W---'` then COUNT isn't even in the top 10.
- of course… if we write a program to do this…. more on that later.

strategy two: have a second guess ready

- alternately, we could just have another guess ready, one that generally speaking cuts down the uncertainty a lot
- so…. guess RAISE and then something else?

good first two guesses

- actually, it doesn't necessarily make sense to guess RAISE first if we're going to decide on two guesses in advance.
- RAISE was the word that minimized the expected uncertainty after one guess, what about the expected uncertainty after two guesses?
- it turns out (after a lot of crunching the data in a python script, the SQL database wasn't able to manage the query) that the best pair of words is…

drum roll…

- TRICE followed by SALON
- why isn't it RAISE and something else?
- well, basically because we're ignoring the response from the first guess. The expected uncertainty after guessing TRICE-SALON 1.592 bits. After RAISE the best next guess (ignoring the response to the first guess) is CLOTH, which comes in at 1.776 bits.
-

should we guess RAISE or TRICE-SALON?

- good question. If you were able to keep a list in your head of the best second guesses after RAISE, depending on the response to RAISE, then yeah, you're better off guessing RAISE
- I can't do that, at least not without a cheat sheet.
- so I do TRICE, usually followed by SALON, unless there's something obvious that pops out after TRICE. And TRICE by itself is not that bad of a first guess, the expected uncertainty after TRICE is 5.54 bits, certainly worse than RAISE (5.30 bits) but it's 23rd on the list of best guesses.

Questions?

strategy one again: minimize expected uncertainty

- as I mentioned earlier, it's possible to compute a second (and future) guess by again trying to minimize expected uncertainty
- specifically, you need to look at the list of remaining possible solutions, given what you know, and find a guess that minimizes the expected uncertainty
- the SQL is very similar to what we used to find RAISE as a good guess in the first place, but with some extra criteria.

a bit of SQL

- let's go back to our first example - our first guess was ABOUT and the answer was (we now know) BLOWN
- the ABOUT guess came back with the response `-WB--`. We want to figure out a guess that minimizes the expected uncertainty, given the fact that the answer space has now been restricted to words that match ABOUT=>`-WB--`
- we have to start by figuring what words those are, which is a simple query:

```
Select answer from scores where guess='ABOUT' and
score='-WB- -';
```

a bit more SQL

- now to find an optimal guess:

```
select guess, sum(c * log(2, c)) / sum(c) from (select
guess, score, count(*) as c from scores where answer in
(...)  group by 1, 2) as t1 group by 1 order by 2;
```

- where the ellipsis should be replaced with the list of still allowable answers
- turns out the two best guesses at this point are LIKEN and MODEL, each with an expected uncertainty of 0.667 bits.
- the word we did guess, BROOD turns out to have an expected uncertainty of 1.083 bits and is only the 47th best guess
- You might say at this point that BROOD at least has a chance of being the right answer and that's true, but the probability of that happening is low

rinse and repeat…

- again look for a list of answers that are consistent with the answer from ABOUT (-WB–) and for LIKEN (W---B). Turns out that the list of consistent words contains… just one word: BLOWN
- so minimizing uncertainty got us to the answer faster than using our intuition and trying to figure out which words it could be - that took us ABOUT, BROOD, BLOCK, BLOWN
- granted that we had an advantage here of actually having the list of allowed answers in a database table… it's still pretty cool

this can be automated

- I wrote a bit of python code to do this repeated work

```
python solve.py blown | jq .
[
  {
    "guess": "raise",
    "expected_uncertainty_after_guess": 5.298886787326121,
    "compatible": true,
    "uncertainty_before_guess": 11.1767964781476,
    "score": "-----",
    "turn": 1
  },
  {
    "guess": "mulch",
    "expected_uncertainty_after_guess": 2.1806654751000516,
    "compatible": true,
    "uncertainty_before_guess": 7.392317422778761,
    "score": "--W--",
    "turn": 2
  },
  {
    "guess": "blown",
    "expected_uncertainty_after_guess": 0.33963482158310493,
    "compatible": true,
    "uncertainty_before_guess": 3.8073549220576037,
    "score": "BBBBB",
    "turn": 3
  }
]
```

and of course…

- I wrote a shell script to try it against all the words in the answer list.
- the results? mean number of guesses to get the right answer is 3.495. Which is probably about as good as most of us do: 3 on a good day.
- if we're being honest, the times we get a 2 it's just lucky that our first guess lines up with the answer
- but 3 or 3.5… yeah, those seem reasonable

does it always finish in 6?

- hell yeah!
- here's a histogram of how it does:

```
   1 1
  61 2
1139 3
1020 4
  92 5
   2 6
```

- the two words that take six guesses are: OFFER and HOVER

what about "hard mode"?

- hard mode means that every guess you make has to be consistent with the information you have so far
- we can add that to our program (another criterion in the SQL)
- the average number of guesses does creep up slightly, from 3.495 to 3.599
- but the tail gets longer - now some words don't get finished in the 6 guess limit

hard mode failures

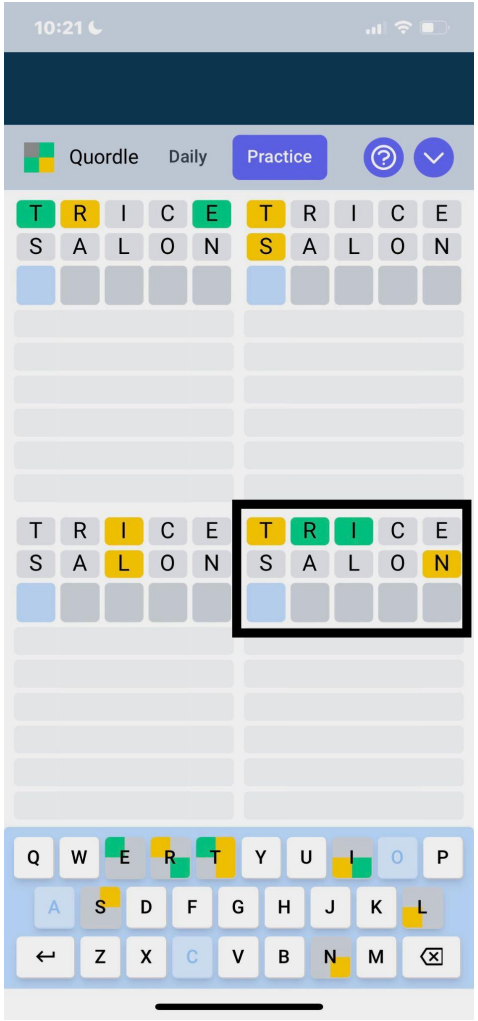
- eleven words ("foyer", "goner", "graze", "match", "swore", "tatty", "waste", "watch", "water", "wight", "willy") take 7 or more guesses.
- why does this happen? Let's see what happens with "GRAZE"
- guesses are: RAISE, GRACE, GRADE, GRAPE, GRATE, GRAVE, GRAZE
- yes, the dreaded blind alley… and if you're playing in hard mode there's no way to get out.
- in non-hard mode, it is solved in five guesses: RAISE, TRACK, PUDGY, GRAVE, GRAZE

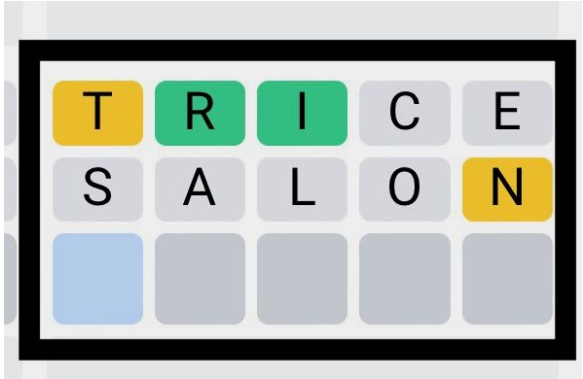
Questions?

Did somebody say "GRAZE"?

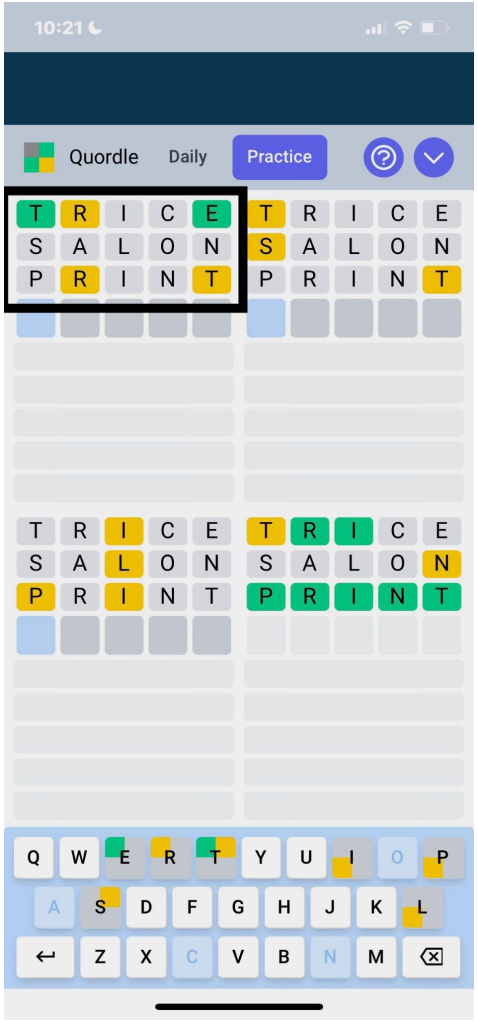
but wait there's more … Quordle

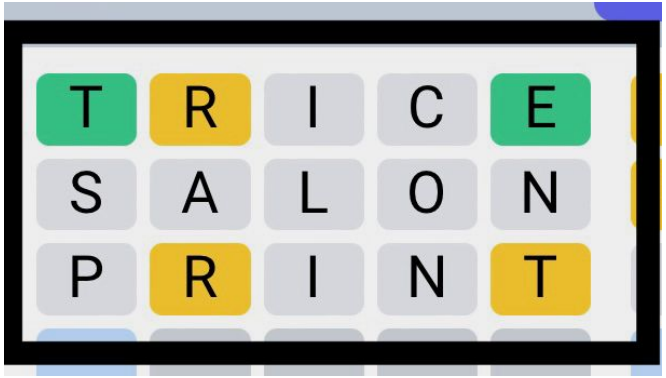
- Quordle is a souped up version of Wordle where you are trying to guess four words simultaneously. I actually like it better than Wordle, it feels like it's less chancy.
- it feels like it should be way harder, trying to do four at once, but
  - you get 9 guesses instead of 6
  - in some ways, it's easier because at each step you can choose which Wordle to focus on at each turn.
- plus there's a practice mode, so you can play more than once a day





- in these screenshots, yellow means "correct in the wrong place" and green means "correct in the right place"
- looking at these four Wordly puzzles, the one at the bottom right looks like `'-RI—'` with an N and a T thrown in.
- surely must be `'-RINT'` and what could that be except PRINT?



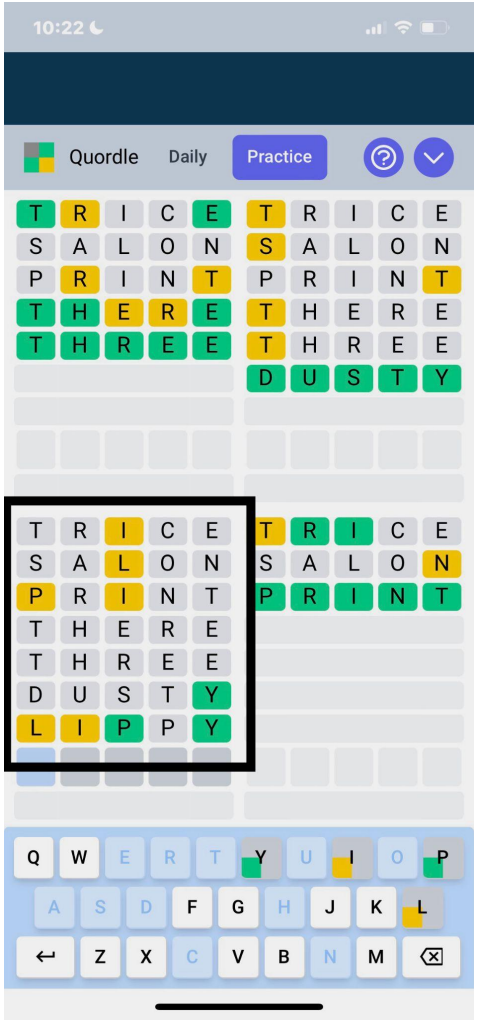


- 'T---E' with an R in there somewhere. Could be lots of things, couldn't it?
- Well, not really… we know there's no vowels except E, U or possibly Y.
- And there's no S so it can't be TERSE
- let's try THERE

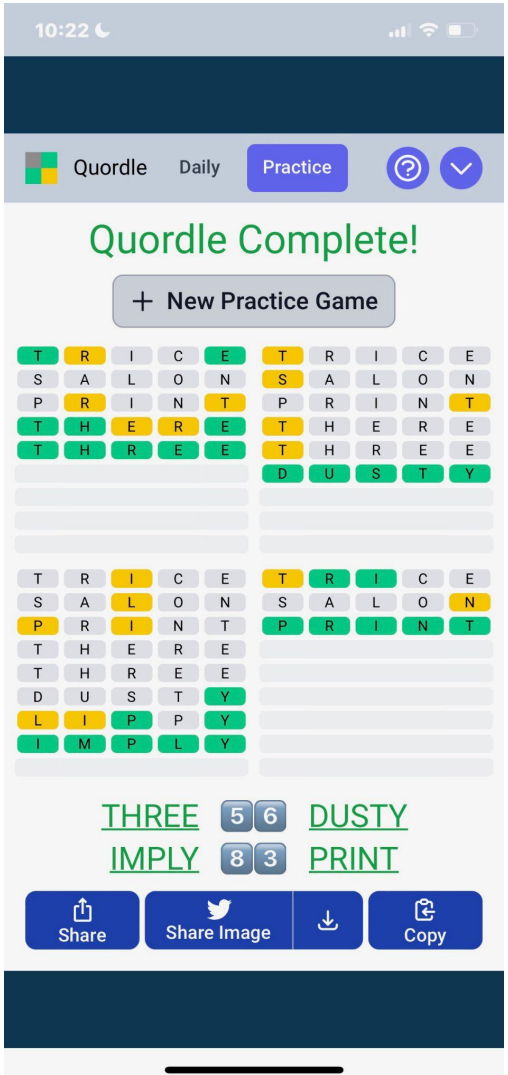
- ah, close but no cigar. Let's fix that

- now this is a curious word. No vowels except for maybe U or Y. Has to have an S and a T.
- I'm thinking something like DUSTY or MUSTY. GUSTY? Is FUSTY a word?
- let's try DUSTY on the hope it will give us info about the other word as well

- we got lucky, it was DUSTy! But luck is an important component of everything, so why not Quordle?
- One word left, it has an L, and I, a P and a Y.
- Gotta be LIPPY!



- oh, guess no. Oops
- I guess it's actually IMPLY



- Yes! 3468… not bad. Certainly I've done worse. Frequently, in fact.
- Occasionally I get 3456 but those are really good boast-worthy days.
- but enough about me. What about Quordle?
- Can we use the same tools we developed for Wordle here?

Quordling with uncertainty

- the key fact that makes the information theory-based tools we've discussed extensible to Quordle is that uncertainty is additive. Remember, that was one of the things we insisted on at the beginning: if you have a pair of independent variables, the uncertainty of the two of them together is just the sum of the uncertainties.
- so computing the uncertainty (and even the expected uncertainty) in the solution to a Quorlde is just the sum of the uncertainty for each of the Wordles that makes it up.
- So yes, RAISE is a great first guess in Quordle, so is TRICE / SALON

Quordling with Code

- Implementing Quordle with code is pretty simple, once you have Wordle working. Basically you just create an array of four Wordles and minimize the sum of the expected uncertainties
- here's how it did on that Quordle: one guess better than I did.

```
$ python qsolve.py three dusty imply print | jq .
[
    {
        "guess": "raise",
        "uncertainty_before_guess": 44.7071859125904,
        "expected_uncertainty_after_guess": 21.195547149304485
    },
    {
        "guess": "count",
        "uncertainty_before_guess": 23.192678098233472,
        "expected_uncertainty_after_guess": 7.1885178424267355
    },
    {
        "guess": "print",
        "uncertainty_before_guess": 7.459431618637297,
        "expected_uncertainty_after_guess": 6.522130377329809
    },
    {
        "guess": "gleam",
        "uncertainty_before_guess": 5.0,
        "expected_uncertainty_after_guess": 0.0
    },
    {
        "guess": "imply",
        "uncertainty_before_guess": 0.0,
        "expected_uncertainty_after_guess": 0.0
    },
    {
        "guess": "dusty",
        "uncertainty_before_guess": 0.0,
        "expected_uncertainty_after_guess": 0.0
    },
    {
        "guess": "three",
        "expected_uncertainty_after_guess": 0.0,
        "uncertainty_before_guess": 0.0
    }
]
```

Multi-quordling

- note that there's nothing special about the number four - we can use the same technique to solve as many Wordles as you like
- `python qsolve.py three dusty imply print anger bingo acute sigma shyly`
- takes a little while but it solves it in twelve guesses
- the number of guesses it takes is just the number of words plus 3, on average

Guessing games in general

- so… what have we accomplished?
- we actually have a general approach for solving any guessing game!
- a guessing game, in this context, is one where there are a finite number of equally likely solutions and the response to a given guess doesn't depend on anything except the guess itself (so, for instance, it doesn't depend on the responses to your previous guesses)
- in fact, we not only have an approach, we have the code: all we have to do is vary the table that has the scores in it.

Example: Wordle in Mastermind mode

- the old Mastermind game was just like Wordle, except you didn't get any information about the location of the letters that corresponded to a particular match. So all you'd get back is the number of letters in the right spot and the number that were correct but in the wrong spot
- (actually it used colored pegs not letters but that doesn't matter)
- and yes, it's harder, because there's more uncertainty after erch guess

How much harder is Mastermind-Wordle?

- we can run through the answer word list checking how well solve.py does against each of them in Mastermind mode, then compare the results to what we got with regular Wordle
- turns out the average number of guesses increases, but not by as much as I had expected - the new average is about 4.60. Recall that (in non-hard-mode, starting with RAISE) the average number of guesses was 3.50. So losing the location data adds just over 1 to the average number of guesses
- for the record - best starting word in Mastermind mode is STARE with an expected uncertainty of 7.94 bits

Wordling in Mastermind mode

- here's a chart comparing Wordling in normal, non-Mastermind (with locations) and Mastermind (no locations) mode. Clearly it is harder to Wordle without the locations
- for instance, there are 61 answers where we can guess the answer in 2 guesses with the locations, but only 8 if we don't have them. But, again, the differences aren't as striking as I had expected, lots of 5's instead of 3's and 4's.

guesses	non-mm	mm
1	1	1
2	61	8
3	1139	115
4	1020	849
5	92	1175
6	2	163
7	0	4

And yes… there's even more

- we could use the same tools to attack ANY guessing game, from Japanese-language Wordle to Mastermind with 17 colors and 15 holes
- we just need the database!
- it's pretty cool that starting with the simple 1 - 10 game that you played as a kid we can get up to understanding much more complex games and see really how much more complex they are.

Questions?

Thanks for your attention… feel free to email me
Rama.Kocherlakota@gmail.com

Code for this presentation can be found at
https://github.com/ramakocherlakota/wordle-level-up-2022/