Lecture 24: Wordle and Information Theory

CS 61A - Summer 2024
Cyrus Bugwadia

Announcements

Announcements

- Scheme is released!
  - Entire project due **Tuesday, 8/6**
  - 4pt extra credit problem included!
- Lab 11 is due **tonight**.
- HW 07 is released and due **Wednesday, 8/7**.
- The final exam is **Thursday, 8/8 from 7-10pm**
- Look out for a final logistics post on Ed soon.

About Cyrus (he/him/his)

- Undergrad @ Cal from 2019-2023 (B.A. in Computer Science)
- Taught 61A 3 times as a TA and 2 times as a tutor
- Industry Experience
  - software engineering internships at Cisco, Intuit, Amazon, Tableau (Salesforce)
  - currently full-time software engineer at Salesforce
- cyrus.bugwadia@berkeley.edu

Contents

1 Wordle / 61Alphabet

2 Representing via Python

3 61Alphabet in the Interpreter

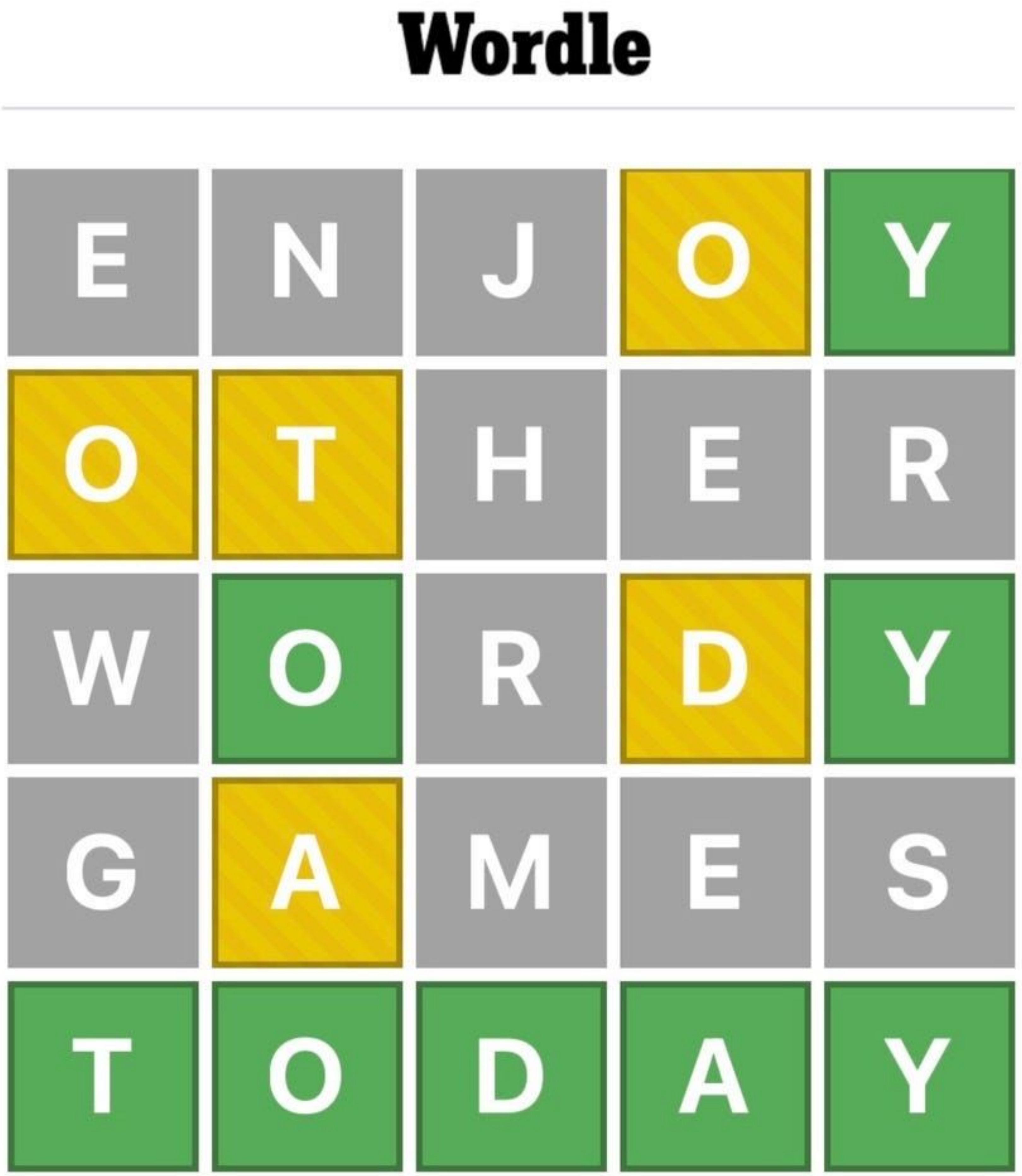
4 61Alphabet Brought to Life! (GUI)

5 Information Theory

1) Wordle / 61Alphabet

Wordle

- New York Times word game
- Player is given 6 guesses to guess a secret 5-letter word
- Each guess gives the player information
  - gray letters: not in the word
  - yellow letters: in the word, in a different spot
  - green letters: in the word, in the exact spot
- Take a few minutes to solve today's Wordle!
  - https://www.nytimes.com/games/wordle/index.html
  - or just Google "NYTimes Wordle"



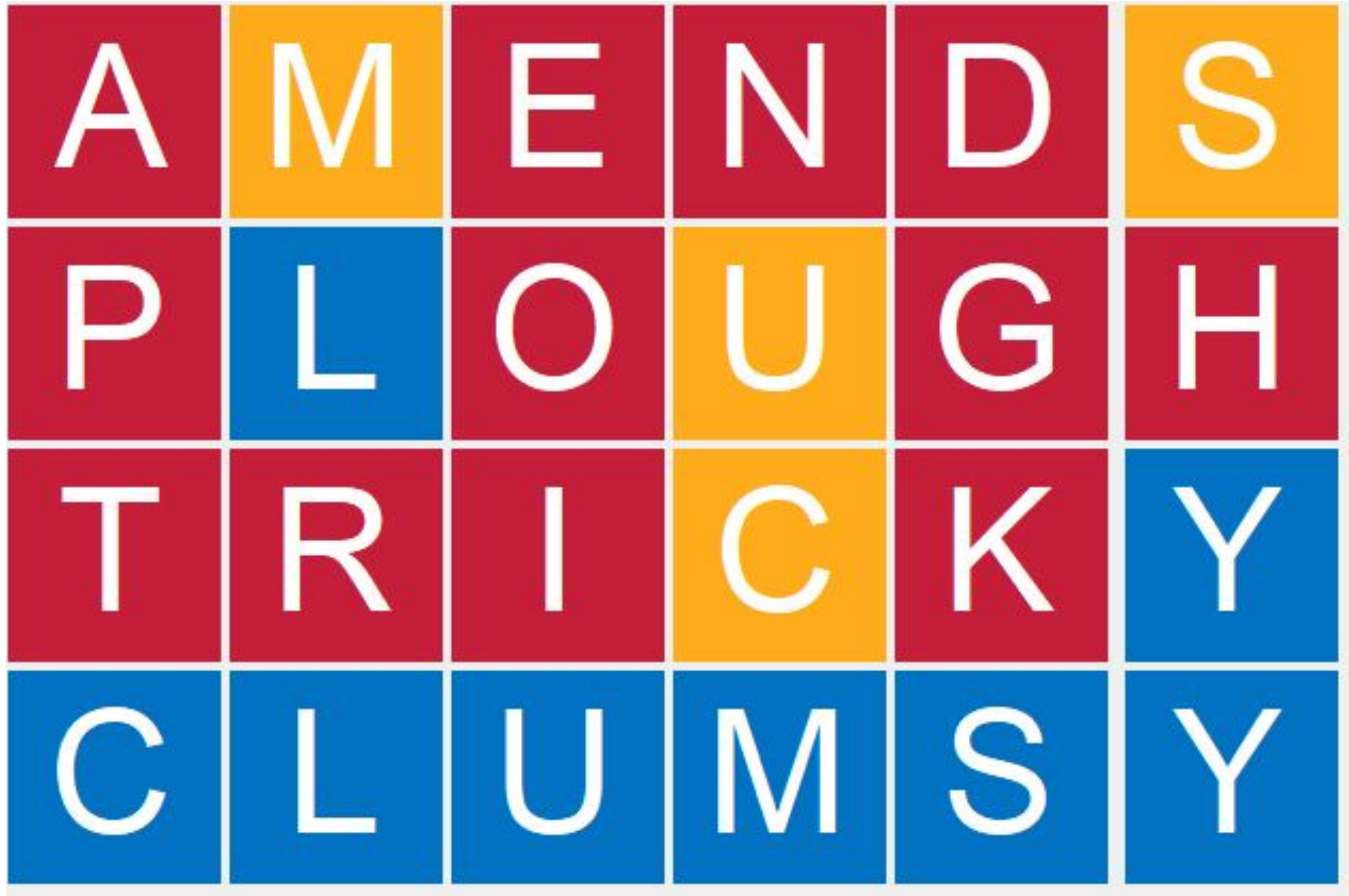
E	N	J	O	Y
O	T	H	E	R
W	O	R	D	Y
G	A	M	E	S
T	O	D	A	Y

More on Wordle

- Secret word changes each day, but is the same for all players worldwide
- Your guess must be an actual 5-letter word (there are ~13,000)
- There are only ~2,300 possible secret words
- Josh Wardle created the game in 2021 for his partner, Palak Shah
  - she hand-selected the secret words from the 13,000
  - eventually released to the public in October 2021
  - sold to New York Times in January 2022 for millions of dollars
- New York Times has trademark rights, so they aggressively take down Wordle clones :(
  - we will work on our own 6-letter variant: **61Alphabet**

61Alphabet (differences from Wordle)

- We will be guessing 6-letter words instead, with up to 7 guesses
  - 3,000+ allowed guesses / secret words
  - words curated manually by me from here
- Color palette
  - 61A Blue for letters in the correct spot
  - Bright Orange for letters in the wrong spot
  - Cardinal Red for missing letters



A	M	E	N	D	S
P	L	O	U	G	H
T	R	I	C	K	Y
C	L	U	M	S	Y

2) Representing via Python

Information / Logic

- How can we keep track of information about a specific Wordle game?
    - use a class! (similar to **`GameState`** in the Ants project)
- What information describes a game?
    - secret word
    - guesses made so far
    - whether the game is won/lost or still in progress
        - can be derived from other info
- How do we process a turn / make a guess?
    - only allow the guess if the game isn't over, and the guess is an actual word
    - add the guess to our guesses list
    - mark correct, misplaced, and incorrect characters as needed

Game class

- Instance variables
  - **`guesses`**: keeps track of guesses made so far
  - **`secret_word`**: the secret word needing to be guessed
- Instance methods
  - **`make_guess`**: makes a guess using the process described earlier, returns whether the guess was valid or not
  - **`guess_count`**: returns the number of guesses made so far
  - **`is_won`**, **`is_lost`**, **`is_over`**: keep track of the status of our game
    - game is won if the last guess is the secret word, lost if too many guesses
  - **`get_row`**: returns the result of a guess (which characters were missing, which were misplaced, and which were correct)

Other Details

- We only want to mark characters as in the incorrect spot if there are enough of them in the secret word, so **`get_row`** needs to keep track of characters that have already been marked correct / misplaced
- We will create global variables **`WORD_LEN`** and **`MAX_GUESSES`** so we can play around with the word length / maximum number of guesses
- Words will be stored in a separate file **`words.txt`**, which we will read from to get our words list
  - when creating a game, we randomly choose one of these words!
- Let's look at our code and play around with it!

3) 61Alphabet in the Interpreter

61Alphabet in the Interpreter

- We should have a way to graphically represent our game
  - we can implement our own `__str__` in the `Game` class! given row data, simply convert each number to the corresponding square emoji
- How do we let the user guess a word?
  - Python provides `input` - a built-in function that returns whatever user enters
- How do we interact with the `Game` class?
  - create a new game
  - loop until the game is finished
    - repeatedly ask for a guess until a valid one is made
    - print out the updated board
  - print out whether we won or lost
- Demo

4) 61Alphabet Brought to Life! (GUI)

GUI

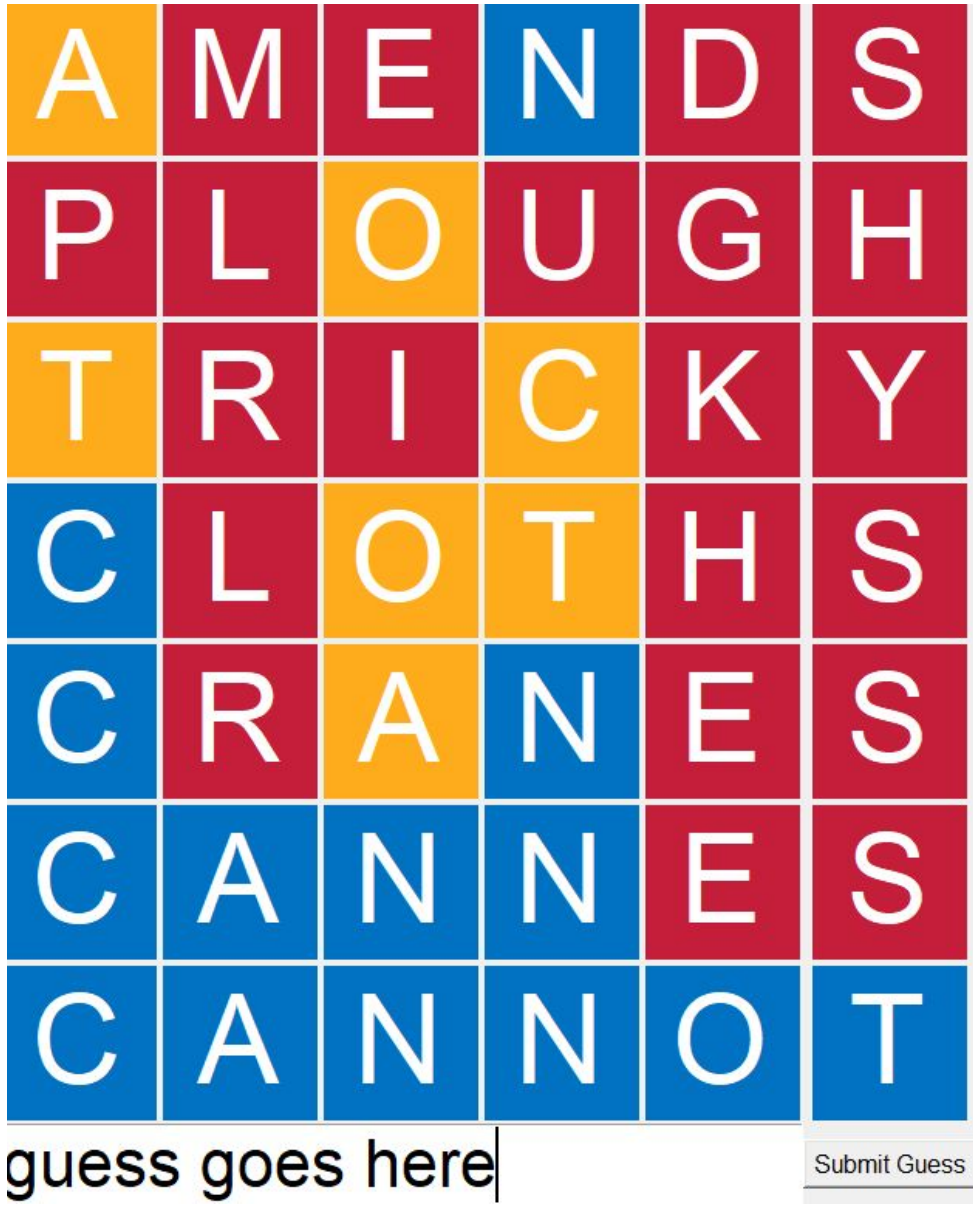
- While interpreter was a good starting point, there were drawbacks
    - limited graphics capabilities, since the interpreter displays actual characters
        - some terminals might not support colors, which we had to use
    - most end-users are used to having an actual graphical interface to work with
- How can we give our game a graphical interface?
    - Python provides a built-in GUI module called **Tkinter** that we can use
    - Tkinter lets us use object-oriented programming to display various objects such as drawings, buttons, text, etc.
    - we can use these to draw the game board, take in and submit guesses, and display messages back to the user (so, everything we need!)

Tkinter

- Classes we need
  - `Tk`: represents the entire window
  - `Entry`: represents text that the user can enter (e.g. guesses)
  - `Button`: represents a button that the user can click (e.g. to submit a guess)
  - `Canvas`: represents a canvas that we can draw on (e.g. board tiles)
- Creating our GUI is a matter of instantiating these classes, and calling methods on them to tell them where to go / what to do

Our GUI Layout

- Tkinter supports many different layouts, but we will use the **grid** layout for our project
    - can be thought of as a grid with a certain number of columns / rows
    - `x.grid(row=r, column=c)` puts **x** in row r and column c
- Row i will display the results of the ith guess
- Column j will display the result for the jth letter
- Below the board, we will let the user enter/submit a guess



A	M	E	N	D	S
P	L	O	U	G	H
T	R	I	C	K	Y
C	L	O	T	H	S
C	R	A	N	E	S
C	A	N	N	E	S
C	A	N	N	O	T

guess goes here

Putting it all together

- Initialize our game and all GUI elements
- Configure GUI elements as necessary
  - make sure elements are in the right spots / use the right font
  - make sure pressing enter / clicking Submit Guess runs the below functionality
- When submitting a guess…
  - do nothing if the game is over, or ask for another guess if it's invalid
  - clear the text entry, and draw a row for the guess
    - color tiles properly / draw letters over them
  - if the game is over (won/lost), notify the user
- Demo

Suggestions

- Play around with the given constants
  - word list, word length, number of guesses allowed are up to your choosing
- Functionality
  - add hard mode support (must use hints)
  - make it daily / save daily progress to a file
- Improve the GUI
  - add separate indicator for which letters are ruled out / misplaced / correct
  - allow for resizing the window
- Add bot functionality (let the computer try to solve the puzzle)
  - more on that after the break

Break

5) Information Theory

Wordle and Information Theory

- Basic idea: the best guesses narrow down the number of possible secret words as much as possible
  - the less words remaining, the closer we are to figuring out the secret word
  - we can consider narrowing down the possibilities as "learning information"
- If we can quantify how much information we learn, we should guess the word that we expect to give us the most
  - this is where information theory comes in!

Bits

- Basic unit of information: the bit
    - a single bit has two possible values: 0, or 1
    - each bit we add means we double the number of possible values
        - e.g. if there are 3 bits, there are 2^3 = 8 possibilities (000, 001, 010, 011, 100, 101, 110, 111)
    - since Wordle has ~13,000 possible guesses, the number of bits of information needed to learn the secret word is given by 2^bits = 13000, or bits = log(13000) ≈ 13.67 bits
        - we consider it cheating to know in advance which 2,300 words are secret words
- Our goal is to make guesses that, on average, give us as many bits as possible
    - each bit of information we gain halves the number of possible words remaining
    - gradle.app: website that grades your Wordle performance based on this concept

Entropy

- **Entropy**: amount of information expected to be gained by learning the result of an event
  - in the context of Wordle, an event is making a certain guess
- How do we calculate it?
  - look at all possible results/outcomes
  - for each result, multiply the probability of that result occurring with how much information we learn if that result actually occurs
  - add up all the values
- How do we calculate how much information we learned?
  - let's say we narrow down the possibilities from 200 words to 10 words
  - this means we have narrowed down the possibilities by a factor of 200/10 = 20
  - 2^bits = 20, giving us bits = log(20) ≈ 4.32 bits

Calculating Wordle Entropy (Example)

- In the context of Wordle, the result of an event/guess is a specific color pattern
- Let's say our possible remaining words are naval, banal, dandy, lanky, pagan, and we want to find the entropy of guessing naval
    - ⅕ chance that the result is 🟩🟩🟩🟩🟩 (if the secret word is naval)
    - ⅕ chance that the result is 🟨🟩⬜🟩🟩 (if the secret word is banal)
    - ⅖ chance that the result is 🟨🟩⬜⬜⬜ (if the secret word is dandy or lanky)
    - ⅕ chance that the result is 🟨🟩⬜🟩⬜ (if the secret word is pagan)
- $\frac{1}{5} * \log(5/1) + \frac{1}{5} * \log(5/1) + \frac{2}{5} * \log(5/2) + \frac{1}{5} * \log(5/1) \approx 1.92$ bits
    - we expect to gain 1.92 bits of information by guessing naval
    - compared to $\log(5) \approx 2.32$ bits needed for the secret word, this is pretty good!

Putting it all together

- Using the technique from the example gives us the entropy for making a specific guess, so we can generalize it to find the best guess overall!
- We consider **all** 13,000 words, calculate their entropies (with all possible **remaining** secret words), and choose the one with the highest entropy
- Repeat the process on every turn, and we have a basic Wordle bot!
  - in real-world testing, this strategy takes about 4 guesses to win on average
- Why consider all 13,000 words?
  - sometimes, the guess with the highest entropy isn't a remaining possibility
  - in the previous example, guessing dingy (which wasn't a possible secret word) had a full 2.32 bits of entropy!
    - this is because it has 5 possible result patterns, 1 for each secret word possibility
  - this is why Wordle's "hard mode" forces you to only make possible guesses

Further Study

- CS 61B features games as projects
- CS 61C goes into how computers represent data / code using bits
    - not really information theory
- CS 70 will likely cover enough probability theory for information theory
    - EECS 126 goes into further detail on probability theory
- EE 229A
    - graduate class on information theory
    - lecture notes by Daniel Raban
- A Mathematical Theory of Communication
    - original 55-page paper by Claude Shannon that describes entropy

Appendix / Resources

- Play Wordle
  - gradle.app to evaluate your performance using information theory
- Download 61Alphabet
- Python Documentation
  - Reading and Writing Files
  - Tkinter
- Wordle Information Theory
  - video by 3Blue1Brown
  - blog post by Oren Bell