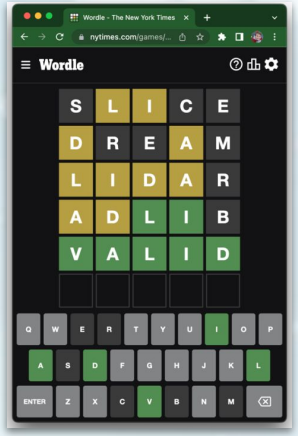
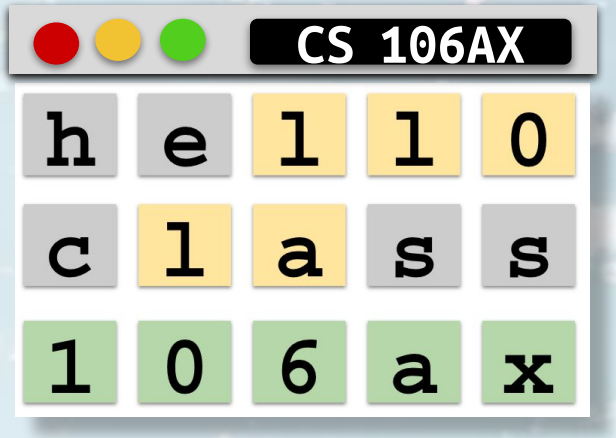
Slides at cs106ax.stanford.edu/assignments.html





YEAH Hours: Assign3

The wonderful Wordle!

CS106AX Autumn 2025

Diego Emilio Padilla, Ben Yan

Stanford | ENGINEERING
Computer Science

Welcome to Week 3/4 of 106AX!

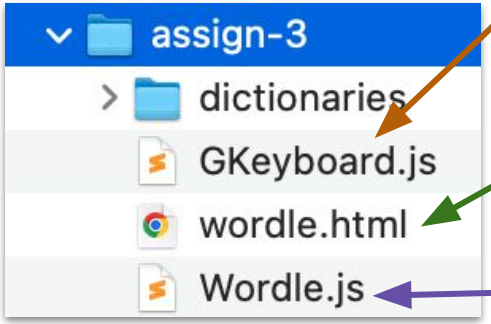
Autumn Week 4 **Winter**

Good luck on any upcoming midterms!

加油, 加油! (add oil, a popular saying)

Downloading Assign3 Code

- Download the **Assign3** code from website
- Then, open the files in your text editor!



Read to understand `GKeyboard` class

Test **Wordle** game here (open in web browser)

Write code here

- When editing the code file, make sure to both (1) **save the code**, and (2) **refresh the browser tab** so the changes are reflected!

Wordle.js

```
/*
* File: Wordle.js
* ----------------
* This program implements the
Wordle game.
*/
"use strict";

// GAME RULES CONSTANTS
const NUM_LETTERS = 5
const NUM_GUESSES = 6;
// SIZING,POSITIONING CONSTANTS
const GUESS_MARGIN = 8;
const GWINDOW_WIDTH = 400;
... other constants

/* Main program */
function Wordle() {
 // You fill this in along with
any helper/callback functions.
}
```

Lecture Recap!

⏳ ~5 minutes

🎶 Once upon a time 🎶

String Methods You May Find Useful

`str.length`

Returns length of string

`str.charAt(i)`, or simply `str[i]`

Returns character of string at **index i**

```
let newStr = str.toLowerCase()
```

"hello world!"

Returns copy of string converted to lowercase.

```
let newStr = str.toUpperCase()
```

"HELLO WORLD!"

Returns a copy of the string converted to uppercase.

```
let substr = str.substring(start, end)
```

Extracts and returns substring between index start (inclusive) and end (exclusive), e.g., "earth".substring(0,3) → "ear"

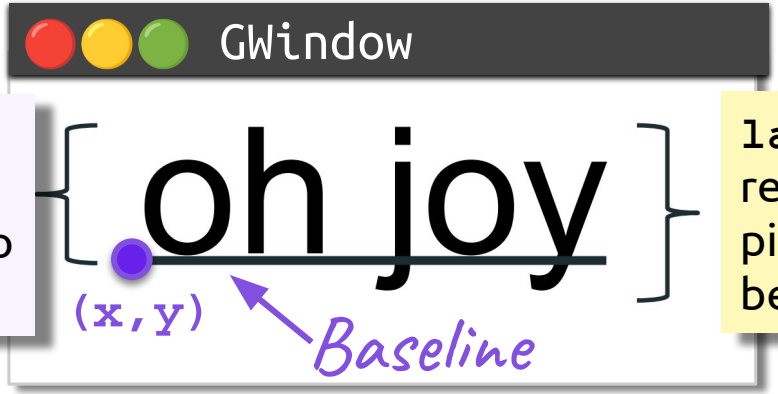
```
str.includes(substr) //returns true or false
```

Checks if `str` includes `substr` as a substring (can be 1 letter) at all

JSGraphics Library: GLabel

Unlike the other **`JSGraphics`** objects we've seen (**`GRect`**, **`GOval`**, etc), the coordinate / reference point of a GLabel is the **left baseline, not the top left!** ❌

```
let label = GLabel("oh joy");
gw.add(label, x, y);
```



`label.getAscent()` returns distance (in pixels) from baseline to top of label

`label.getHeight()` returns full height in pixels, including parts below the baseline

🎯 Centering and Stylizing GLabel

To **center a `GLabel`** at a coordinate **`(x,y)`** horizontally and vertically, one could perform some neat calculations / geometry with **`.getAscent()`** and **`.getWidth()`**, but alternatively, **convenient methods include:**

`label.setTextAlign("center")` will center it horizontally at the x-coordinate

`label.setBaseline("middle")` will center it vertically at the y-coordinate

From lecture, to stylize a **`GLabel`**, e.g., set the color, font size, etc., we can use

- ❏ `label.setColor(color);` // e.g., "green", "red", "#FFFFFF"
- ❏ `label.setFont(font);` // e.g., "700 36px HelveticaNeue"

Happy to discuss any questions!

🎮 *Next Up: Overview, trickier bits of Wordle!*

Wordle Constants

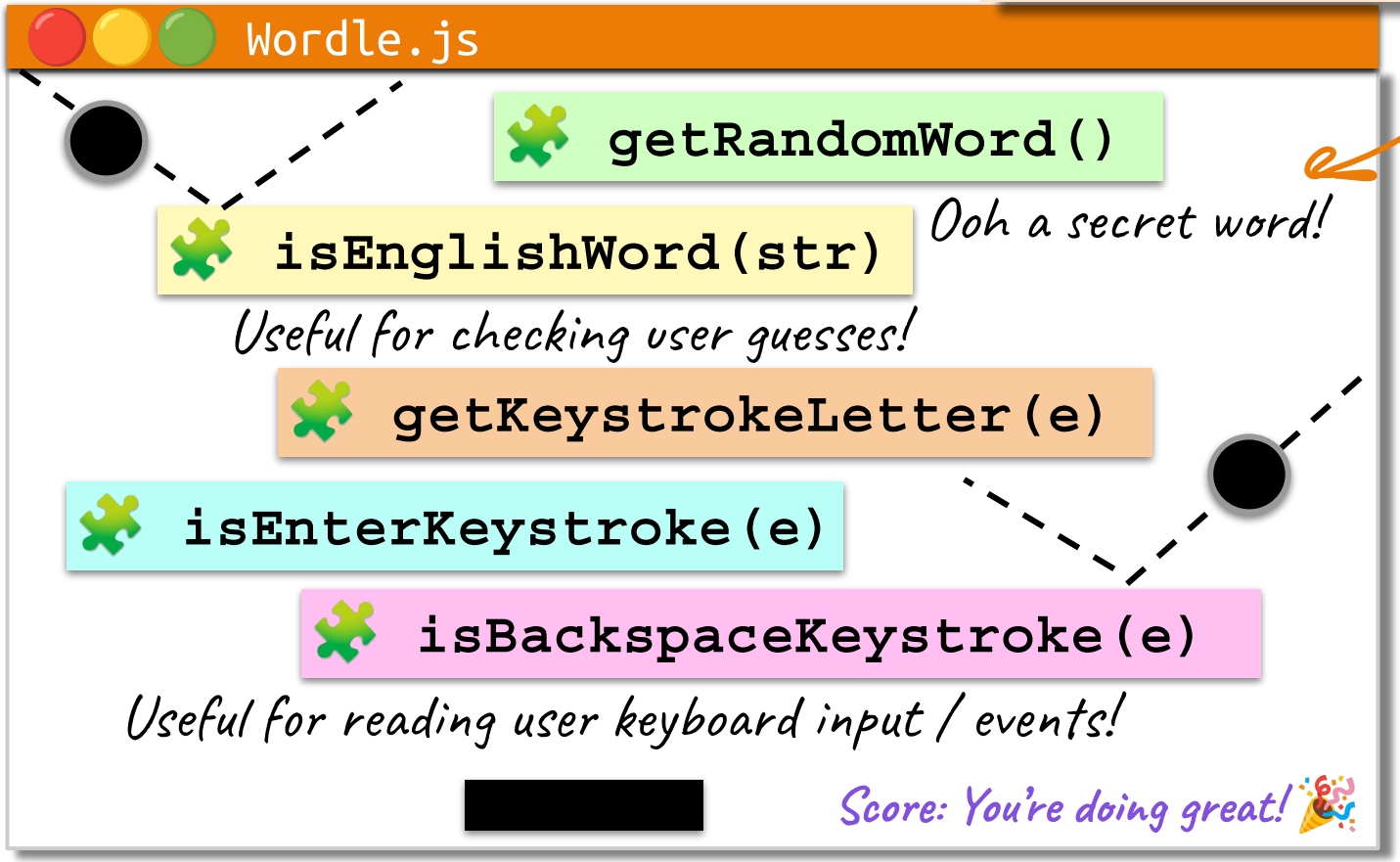
These are already defined in the file for you, and we encourage using them!

Wordle.js

```
// GAME RULES CONSTANTS
const NUM_LETTERS = 5;  // The number of letters in each guess
const NUM_GUESSES = 6;  // The number of guesses the player has to win

// SIZING AND POSITIONING CONSTANTS
const SECTION_SEP = 32; // The space between the grid, alert, and keyboard sections
const GUESS_MARGIN = 8; // The space around each guess square
const GWINDOW_WIDTH = 400;  // The width of the GWindow

// The size of each guess square (computed to fill the entire GWINDOW_WIDTH)
const GUESS_SQUARE_SIZE =
 (GWINDOW_WIDTH - GUESS_MARGIN * 2 * NUM_LETTERS) / NUM_LETTERS;

// STYLISTIC CONSTANTS
const COLORBLIND_MODE = false; // If true, uses R/G colorblind friendly colors

// Background/Border Colors
const BORDER_COLOR = "#3A3A3C"; // Color for border around guess squares
const BACKGROUND_CORRECT_COLOR = COLORBLIND_MODE ? "#E37E43" : "#618C55";
const BACKGROUND_FOUND_COLOR = COLORBLIND_MODE ? "#94C1F6" : "#B1A04C";
const BACKGROUND_WRONG_COLOR = "#3A3A3C";
```

Wordle Helper Functions

Some are **already written** for you, and we encourage using them!



NYT Wordle Game (by Josh Wardle)

6 tries to guess a 5-letter word
(players know, can be quite tough!)

When each guess is submitted, the letters light up as follows:

E GREEN means letter in **correct** place

L YELLOW means letter in a **different place** in the word

O GRAY means letter **not in word** at all

🎹 As we ~~slay~~ play, to help us out, the keyboard also lights up with these `green/yellow/gray` hues!

Implementation as Trilogy: Parts I, II, III

I mean, who doesn't like a trilogy?
(when done well of course)

As such, we introduce the Assign3: Wordle ~~tragedy~~ trilogy!

106A/AX	106B	107

1 **A New View** – You'll create the static display for Wordle, i.e. grid of guess squares, adding the keyboard, etc.

2 **Breakout! Strikes Back** – Like Assign2, you'll design the game state, e.g., store previous / current guesses, what else?

3 **Return of the UI** – the force of destiny (your keyboard) unites game state and UI controls to defeat Darth Wordle

Part I: The View – Write a draw() function

The first part is creating a static display of the Wordle game. Our cast includes:

The **`GWindow`** that everything resides on

The grid of guess squares (one word per row, one letter in each cell). Starts off as all empty.

H	E	L	L	O
W	O	R	L	D
I	M			

A place to put a game alert (if any)

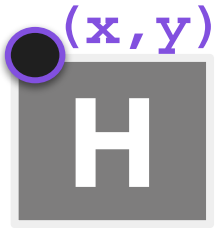
You won! You lost! :(Not a valid guess!

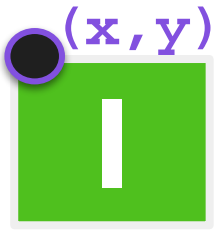
The GKeyboard defined in **`GKeyboard.js`**

GKeyboard(x, y, width, textColor, defaultColor);

Decomposition: Guess Square, Row, Grid

Guess Square



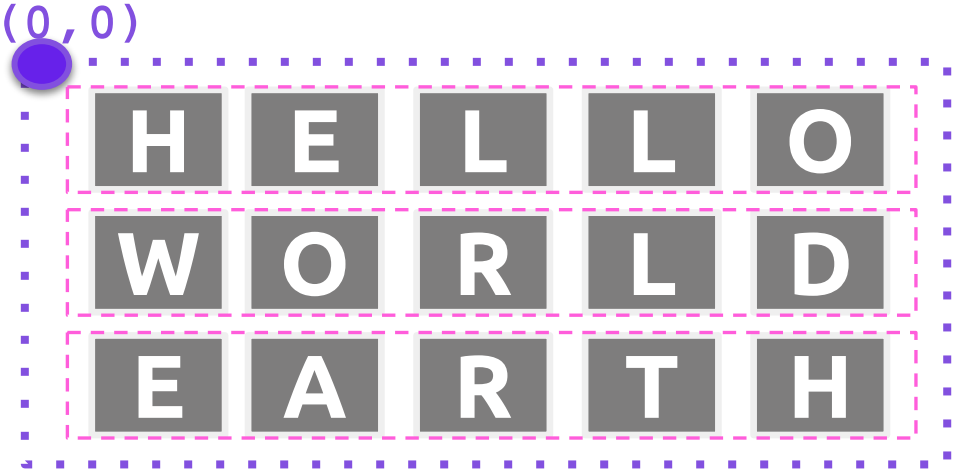


A **GCompound** with a square and GLabel. Coordinate (**x,y**), color, letter are customizable.

Guess Row

A **GCompound** of several **Guess Squares** in a row.

Guess Grid

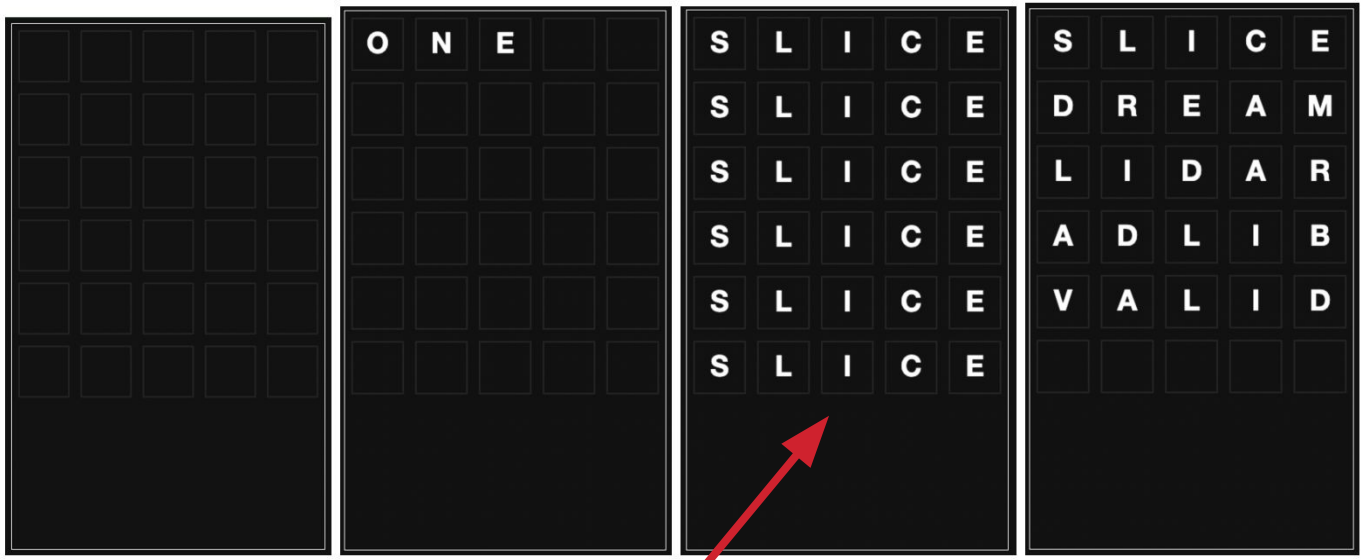


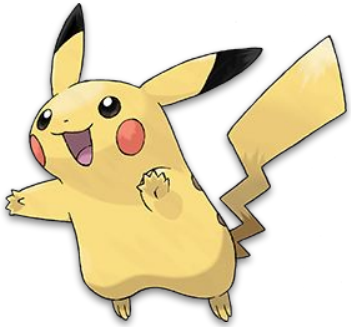
A **GCompound** of several **Guess Rows** that form a grid of words.

It may be helpful to have a helper function for each!

Decomposition: Guess Grid

By changing parameter(s), e.g., different words, to a **`produceGuessGrid`** function, your program should able to draw the following:





You don't have to worry about highlighting the squares the right color here! (that's for the middle of Part II)

🧠 Part II: Program State (longest part imo!)

Your primary goal here is to create **the state / brain of the program i.e. what it should keep track of** as the Wordle game is being played?

secret word: J E R R Y

Should be randomly chosen at start! May want to `console.log()` for easier debugging :)

guesses:

what user has typed in

["ROBLE" , "TOYON" , "OK"]

"ROBLE", "TOYON": Previous, submitted guesses

"OK": Current guess

Multiple possible ways to represent state! 🎨

Also, to think about: do we need anything else? 🤔

These should all become variables placed at the **very start of the top-level Wordle function!** They have 💖 **main character energy** 💖

Part II: Program State

Now, the draw() function should **reflect the state** of the program, e.g., **populate the guess grid** with what the user has typed in so far,

secret word = "EARTH"

↳ prob best not to show this on-screen :)

guesses = ["HELLO", "WORLD", "DREAM", "CATCH", "EXO"];

… anything else that's apart of your game state

GAME STATE

draw()

Next Up: Highlighting Letter Colors!

Part II: getColor(guessWord, letterIndex)

Now, given a secret word + guess, how do we **highlight the guess's letters?**

- ❏ This can get tricky in some **yellow-square / duplicate letter cases**, so I would recommend writing out examples – and playing with the actual Wordle game!

✅ **Right letter, right place → Green**

secret word	J	E	R	R	Y
guess	B	E	R	R	Y

❌ **Letter not in word at all → Gray**

secret word	J	E	R	R	Y
guess	B	E	A	R	S

Letter in word, different place → Yellow (see below)

secret	T	R	A	I	N
guess	R	A	I	N	Y

	R	E	A	R	S
	R	I	V	E	R

😰 **Unless all letter copies found by then → Gray**

secret	T	R	A	I	N
guess	R	E	A	R	S

only 1 **R** to find, so 2nd gray

	W	O	R	L	D
	H	E	L	L	O

also only 1 **L** to find

Edge Case Hint: For letter **guessWord[index]**, **how can we track the # of copies we still need to find in the secretWord?** If we still have copies left to find by then, yellow, otherwise gray.

Part II: Letter Color Highlighting

One thing to check: **which words / letters on-screen should be highlighted?**

secret word = E A R T H

guesses = [“HELLO”, “WORLD”, “DREAM”, “CATCH”, “EAR”];

… anything else that’s apart of your game state

🧠 **GAME STATE**

Part II: Letter Color Highlighting

One thing to check: **which words / letters on-screen should be highlighted?**

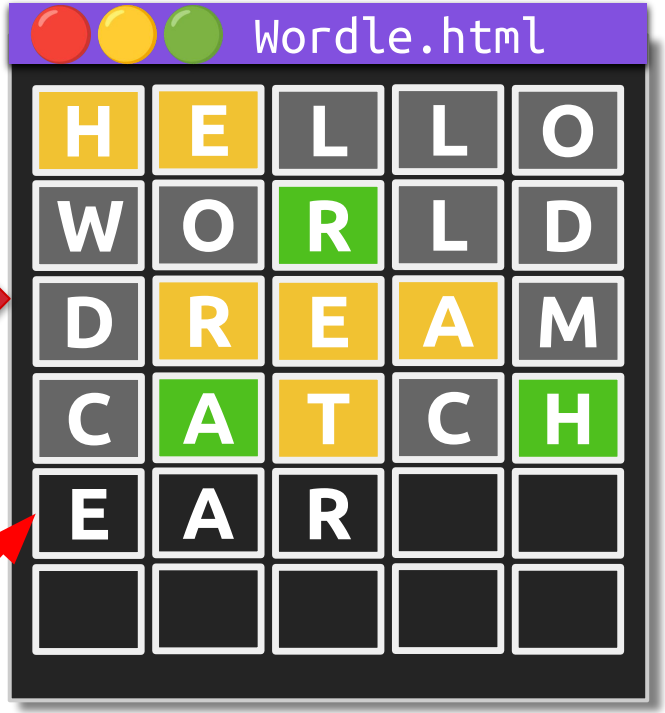
secret word = E A R T H

guesses = [“HELLO”, “WORLD”, “DREAM”, “CATCH”, “EAR”];

… anything else that’s apart of your game state

🧠 **GAME STATE**

draw()

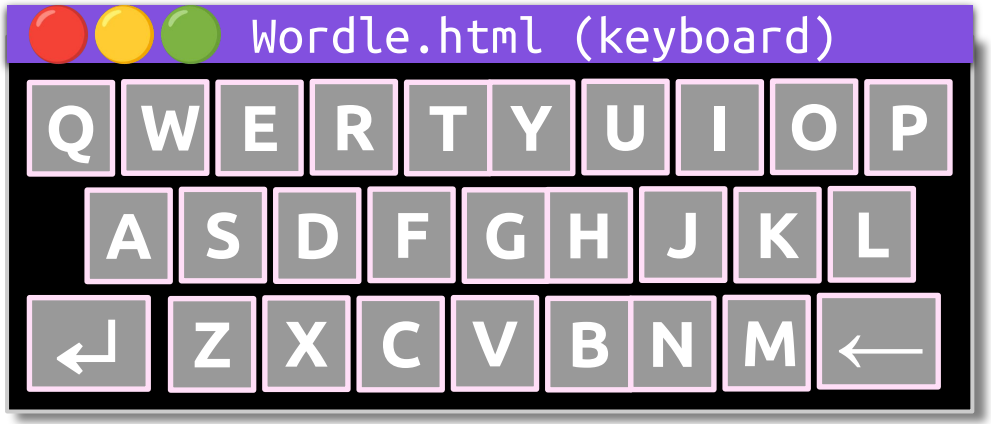


🤔 Should the letters of the **current guess / guess-in-progress** be highlighted in color?

Part II: Keyboard Color Highlighting

The keys on the keyboard should also be highlighted with same color scheme!

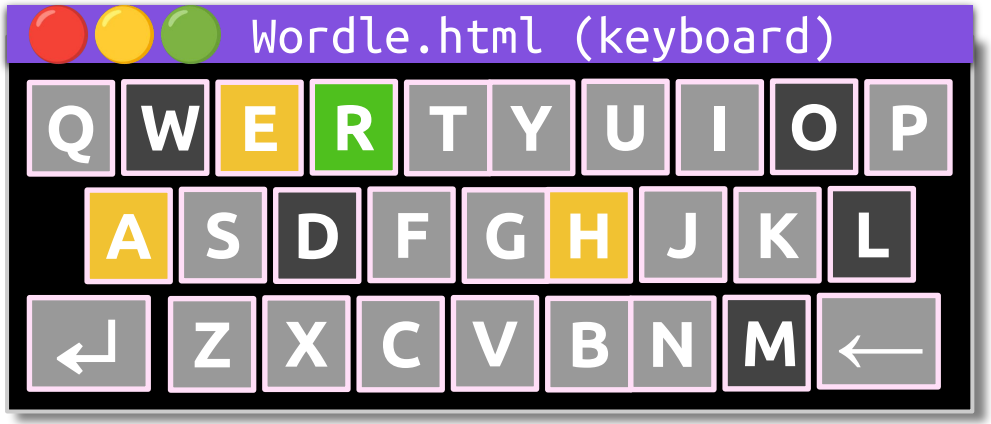
- Use `keyboard.setKeyColor(keyLetter,color)` method
- Unlike letter highlighting, depends partly on prior guesses



Part II: Keyboard Color Highlighting

The keys on the keyboard should also be highlighted with same color scheme!

- Use **`keyboard.setKeyColor(keyLetter,color)`** method
- Unlike letter highlighting, depends partly on prior guesses
- **Tip:** Get a list of keys that need to be painted green, yellow, and gray (may be overlap), and paint them in a way so that green > yellow > gray in precedence



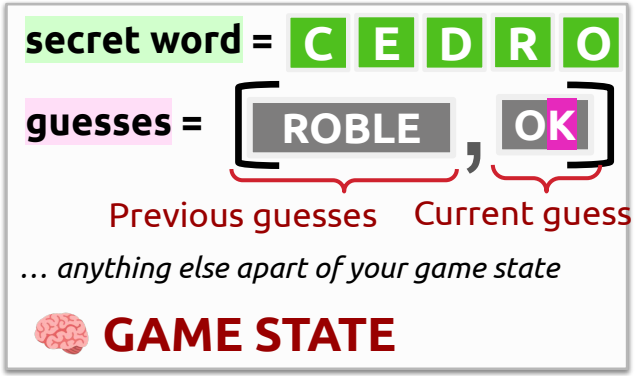
*Here, the letter **R** is still Green (Guess #2) even though in the latest guess, it was Yellow.*

Part III: Keyboard Interactivity / Gameplay

🕹️ Model-View-Controller (MVC) Model

- **Key Idea:** When game state changes via user actions, the view should also
- We **model** the game application via the state, which determines the **view**
- User interacts with **controls**, e.g., pressing **K**, which updates the **relevant state** (current guess), which in turn **redraws the view** (new letter on-screen)

1. Controls – User presses key



2. State — Change state to update guess-in-progress

3. View — Call draw() to reflect new state / letter

Part III: Keyboard Interactivity

Please read the **`GKeyboard.js`** file for more in-depth documentation! 📝

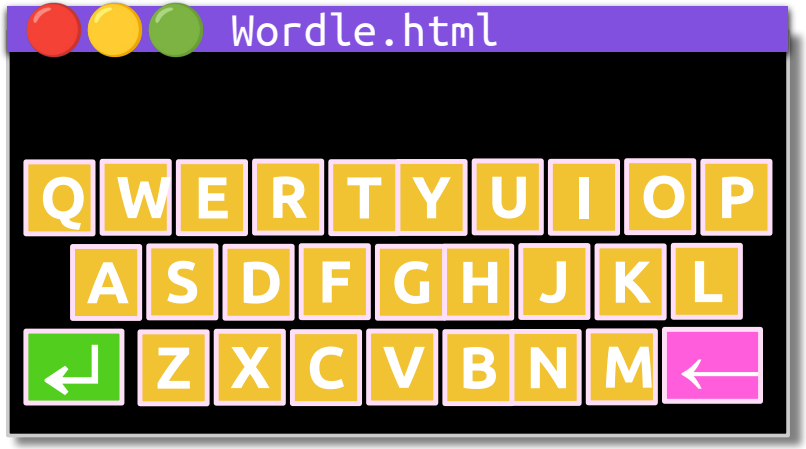
To handle and respond to interactions with the on-screen keyboard, we use **custom GKeyboard event listeners.**

Like mouse events, on-screen **keyclick events are affiliated with a parameter e**, which can inform us, e.g., what letter key was pressed!

These listeners – which we will write our own **event-handling functions** for – are:

- ❑ **`"keyclick"`** **`[Q]`**
- ❑ **`"enter"`** ↵
- ❑ **`"backspace"`** ←

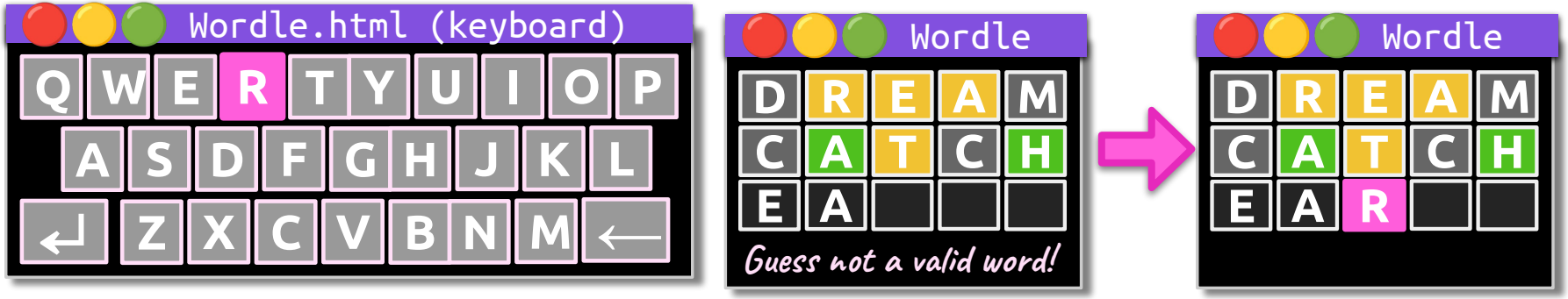
Like before, we use **.addEventListener()** to mount & activate the event listeners 🔊



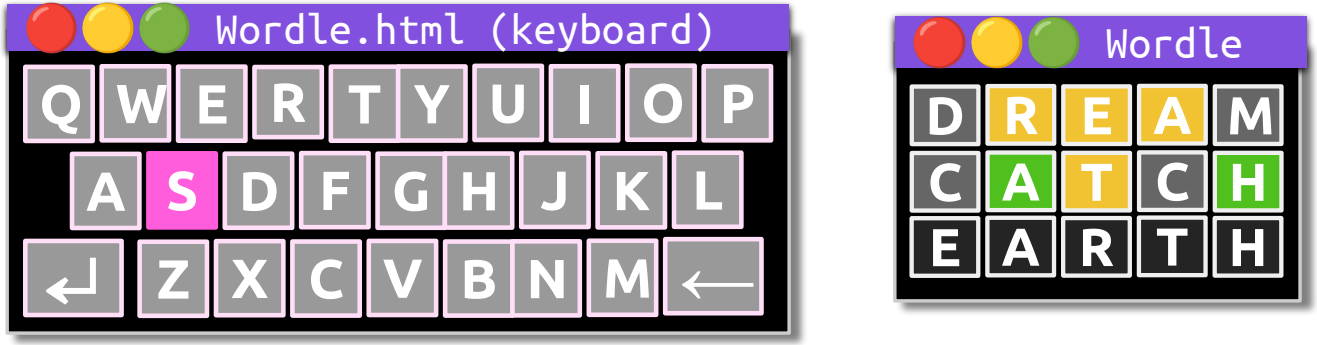
Part III: `keyclick` Event

Think about how you can change state to make this happen!

Should add the letter on to the current guess in-progress, clear any alerts

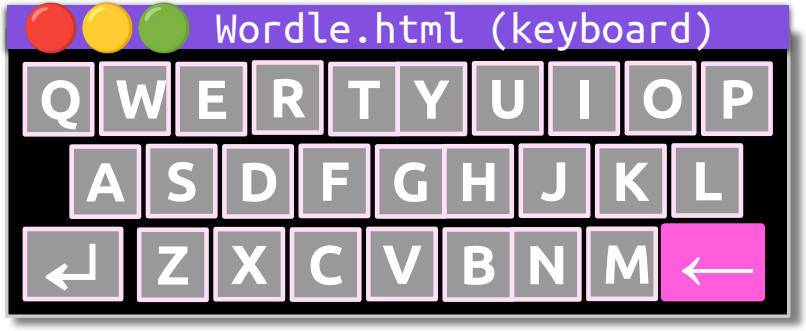


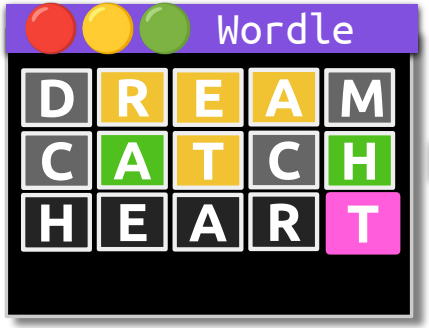
Case to Consider: What happens if the guess is already full / **N_LETTERS**?

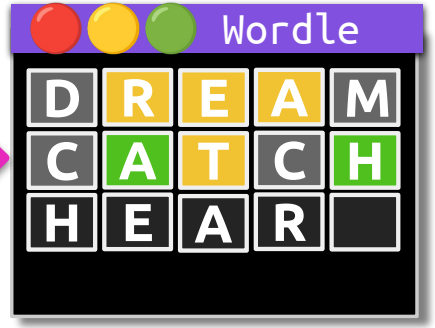


Part III: `backspace` Event

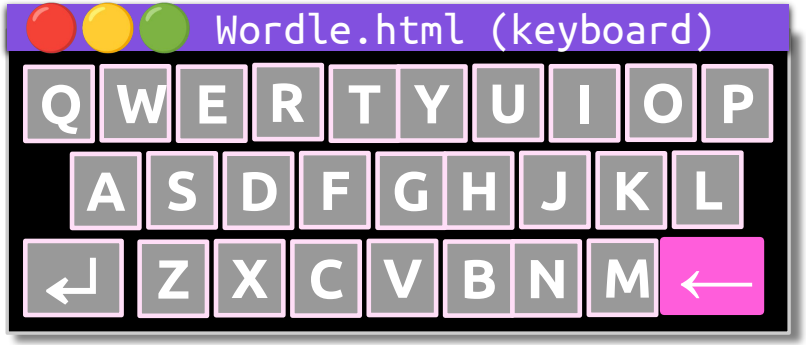
Remove a letter from their current guess-in-progress

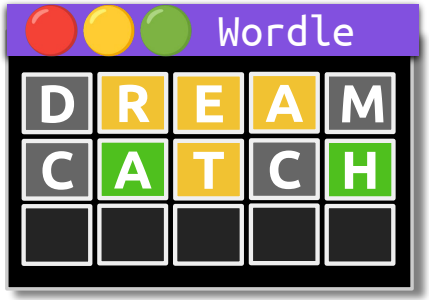






🙄 **Case to Consider:** What happens if the guess is already empty / length = 0?

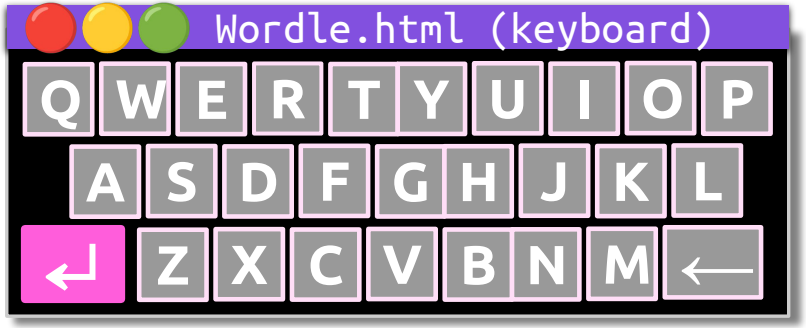


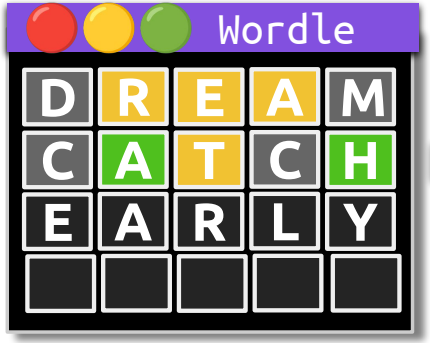


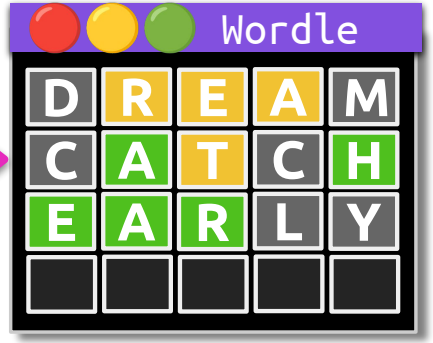
⏩ Part III: `enter` Event

How can you change state so that our submitted guess's letters now get highlighted?

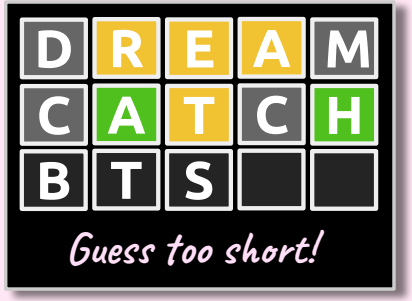
Submit user's guess if valid, begin accepting new guess if game not over



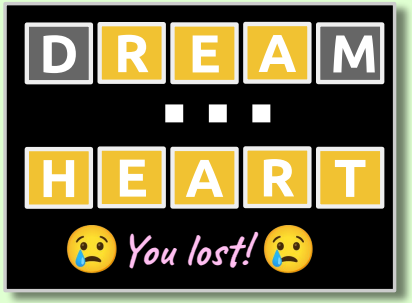




Alert user and don't accept the guess if not a valid English word or too short



If user won or lost the game, notify them – and don't accept any new events (**game over**)



Part III: Lastly, the Physical Keyboard!

Getting the **on-screen keyboard** to work (i.e. handling keystroke, backspace, and enter clicks) is 90% of the work! Finally, we'll **wire in the real, physical keyboard**.

We have this already!

Event-Handlers

For responding to `keyclick`, `enter`, `backspace`

We want this bridge also!

The **`GWindow`** responds to general physical **keydown** events **e**; we can use the provided

`getKeystrokeLetter()` `isEnterKeystroke()` `isBackspaceKeystroke()`

helper functions determine which **event-handling callback function** to use :)

Final Tips and Recs (Similar to Breakout!)

- Write and test your code incrementally
  - **Set a milestone schedule**, knocking them down over the course of the week
  - Ensure each part is working before moving on
  - Fifteen-page (detailed) handout, start early!
- 🧩 **Decompose early!** The graphics / algorithms are noticeably more complex than Breakout!, so this can simplify the work considerably :)
- Come see us in **office hours** if you have questions, get stuck, or want testing tips!
  - Test and test! It's **playing a game** :)
  - **Play the actual game of Wordle to verify!**

Sample Schedule

Sun: Read handout, go to YEAH! :) Get as much of Part I done as possible

Mon: Start Part II

Tues: Work on Part II (toughest part imo)

Wed: Lock in for Math

Thurs: Post-math midterm, pull up to OH for Part III / group emotional support

By Fri: Submit! Relax

Best wishes on midterms! :)

Please feel free to reach out after YEAH or in office hours if you have any questions!

Last Monday, October 6th
= Mid-Autumn or Mooncake Festival!