-- INTRO --

(Camera close up on Word Jumble title screen)

Hi guys,

This is the same circuit that I used in my last video, but by uploading new code, I've turned it into something completely different.

-- DEMO --

(Camera focuses on newspaper version with Arduino in the wings)

In case you're not familiar with Word Jumble, it's a word puzzle that is commonly found in newspapers.

In case you're not familiar with newspapers, these are paper version of websites.  No videos or animated GIFs, just static images and text.

These words have been jumbled and you're supposed to un-jumble them.

When you have unscrambled all the words, you need to unscramble these circled letters into the final answer which is usually a clever pun.

My Arduino version only implements these and not this portion.

(put newspaper aside, bring Arduino into focus)

To start the game, you press this button and it will randomly pick a word and scramble it for your enjoyment. :-)

The game comes with 100 words, I'll show you how to put more later.

To solve the puzzle you drag and drop letters, but how do you do that without a mouse or a touch screen?!  Well, what we have here is a one button one dimensional mouse!  So you can only move in the X axis, but that's all we need.

And if you hold the "mouse"  button, you can drag the letter around and drop it where ever you want.  Isn't that cool?!

I'll explain the code later, but let's solve this puzzle so I can show you what happens when you solve it.

(Solve it)

No matter how long or short the word is, the knob always moves the "pointer" the same way.  leftmost turn points to first character, and rightmost turn points to the 16th character.  That way the "feel" of the mouse remains consistent regardless of the word length.

I also put checks in the code so you can't drag letters out of the word boundaries, that way you could easily slam letters to the beginning or ending of a word.

(solved) \*\*\* GOOD JOB! \*\*\*

You can press the button to start a new game.

-- SUMMARY --

If you're not interested in the HOW part of this video, thanks for watching!

Bye!

For the rest of you, we'll talk about:

- LCD Custom Graphics (this arrow is a custom graphic)

- map() function (How I use the map function to implement the drag and drop thing)

- sizeof() function (How to use sizeof() to dynamically compute array lengths)

- PROGMEM (How to use PROGMEM to store more data when you run out of memory)

-- TITLE SCREEN --

Word Jumble

Arduino Starter Kit project #3

HariFun#176

-- CONTENT --

As usual, all my code is available on github. Link is in the video description on YouTube.

While you could connect these modules directly to the Arduino, that would require using four or eight Arduino pins (show screenshot of parallel wiring).  A simpler option would be to use a backpack module like.  This module turns the display into an I2C device.

I2C only require CLOCK (point to SCL) and DATA (point to SDA) -- costing us only two Arduino pins instead of four or eight!  Freeing Arduino pins for more fun!

As I mentioned in the unboxing, I bought my Arduino Starter kit from AliExpress in China, so not surprisingly, it comes with with ZERO documentation.  Thankfully, the Arduino is such a popular platform, not only I was able to find documentation, I found several libraries to choose from as well.

My kit came with this I2C LCD backpack, unfortunately, it arrived DOA, so I had to replace it with this AdaFruit backpack. This one also does SPI in addition to I2C, but I'm using the I2C so it should work when my replacement I2C backpack modules arrive from China (show screenshot with price).  Isn't it insane that they could sell these for 83 cents each including shipping?! I wonder what's the cheapest thing anyone ever bought with free shipping?

Arduino comes with hardware I2C on Arduino pins A4 and A5. A4 is Data, and A5 is clock.  The I2C standard require pullup resistors on these two pins, but thankfully the backpack already have those built into them.

Then just supply ground and +5 and we're done with wiring the LCD.  MUCH easier than 8 data pins, command/data select, reset, bla bla bla.

I2C is a protocol where multiple devices could share the same two lines. To communicate with a particular I2C device we need to know its address.  Usually this is in the documentation, but if you don't have it, you could run this bus scanner sketch that would just try to all the various address and see which one responds.  Aha, so ours is XXX.

Let's see if we could do a Hello world.

No problem!

Quick review: Here's where we include library, here we instantiate LCD variable with the address of the I2C backpack, and from there we just use the LCD variable to print stuff.

ok, now that we got the screen working, let's learn how to create custom graphics.  Unlike modern displays where you could control every pixel, these displays are character based. It's made up of two lines of 16 characters each.

But they support custom characters so instead of the usual letters and symbols you can define what a character look like.

You could define up to eight different custom characters, for this game I only needed two: an UP arrow and an underline.   
I knew it was possible, but I didn't know how to do this, so I googled lcd custom character editor (and you can see we're not the first one here:-)).  I like this one,

<https://seetron.com/apps/app_cceditor.html>

so, just draw your custom character and it converts those bits into eight bytes that we plug in into the sketch.

The command to define a custom character is:

<https://www.arduino.cc/en/Reference/LiquidCrystalCreateChar>

*lcd*.createChar(num, data)

The first parameter is which one of the eight custom characters you want to define, the second parameter is the array of eight bit patters we got from the editor.

Then every time you print ASCII zero you see your custom character.

(show custom character).

(show underline)

Cool.

Let's move the arrow around using this potentiometer.

One side of the pot is connected to ground, the other to +5 and the middle wiper is connected to Analog Zero.

As you can see, we get a value of zero at one end and 1023 at the other extreme.  But we want the arrow to move from position zero (point to leftmost) through 15 (right most).  Arduino has a really handy function to map a range of values to a different range of values. The function to do this mapping is aptly named map() (show the docs)

<https://www.arduino.cc/reference/en/language/functions/math/map/>

map(value, fromLow, fromHigh, toLow, toHigh)

So in our case, we want the value returned by analogRead, that value ranges from 0 through 1023, and we want to map that range to 0 though 15.

Easy as that!

So, I kept a variable to keep track of the last "mouse" position, and if it changes, we update the screen.  This way we the display won't be flickering because we only refresh it when the value changes.

Next, let's change what we're moving around depending on whether the button is being pushed or not. (explain)

// print puzzle

// drag letter

ok, now that we got one word working, let's add more words.

Let's just turn this into an array, and pick one at random everytime we reboot.

uh oh... what's this?!

How lame that we can't even have XYZ words?!

So, the Arduino only has 2K of RAM, but it has 32K of Flash memory. That's where the skech or program is stored.

Normally all variables occupy that 2K RAM, that's why we ran out of memory.

Notice that we still got TONS of flash memory!

What we need is a way to tell the Arduino to store our word array in program memory.

PROGMEM.  Hey lookee here! :-)  
So by adding PROGMEM modifier, this variable would be stored in program memory that we have so much of.

Unfortunately, that only puts the pointer to our strings into progmem, the actual strings are still in RAM.  To move the strings into progmem we need to surround EACH string with the F function.  Very tedious, so I used excel to concatenate the needed string for me.

Let's see how much memory we have left now.

As you can see, you could add lots more words if you got tired of these words.

//-- REVIEW --

//So, in this video we learned how to create custom characters, how to use map //and progmem.

Please leave comments if you find this interesting or if you have questions.  I'll do my best to clarify.

Don't forget to like, subscribe, and share. Till next time guys! Bye!

-- END CREDITS --

First, sorry if I wasted your time in the last video.

I did a terrible job explaining the trick and a lot of people didn't understand it.