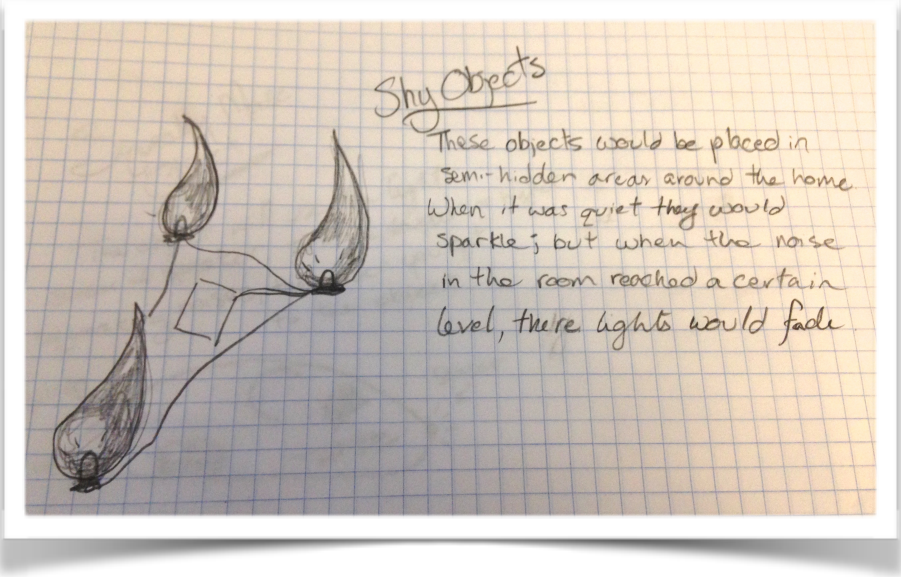
NYU Polytechnic School of Engineering
Programming Physical Objects
Student: Helen Carey
March 3, 2015

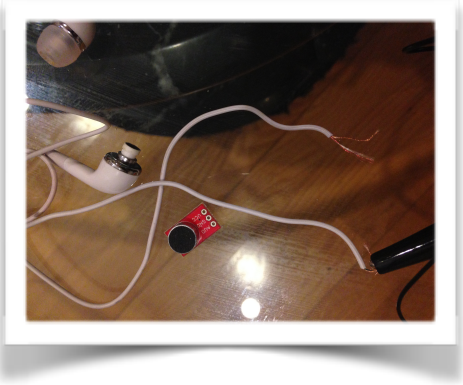
Stupid Pet Tricks 1:

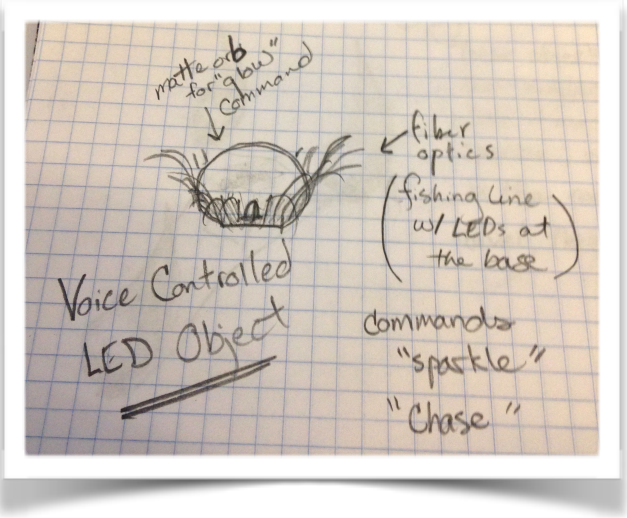
Squeaky Mouse

Exploratory Phase

Initially I had hoped to make an object that could respond to speech. I had seen several tutorials on Instructables.com and youTube, and even purchased a tiny mic + amp (photo right).







Sketch 1 (above); Shy Objects

Sketch 2 (left): Voice Controlled LED Lamp-Object

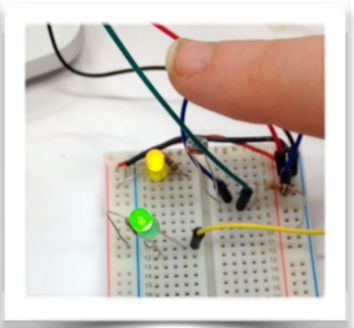
However, I soon learned that a voice recognition project would be beyond my abilities and price range at this point. The voice recognition projects that I could understand either used BitVoicer software, which is only available for PC, or required an additional shield. (There are voice recognition programs for Mac that do not require an additional shield, notably uSpeech, but they seemed less developed and required some knowledge of computational phonetics.)

I decided to be content with simply making a little noise with the piezo instead.

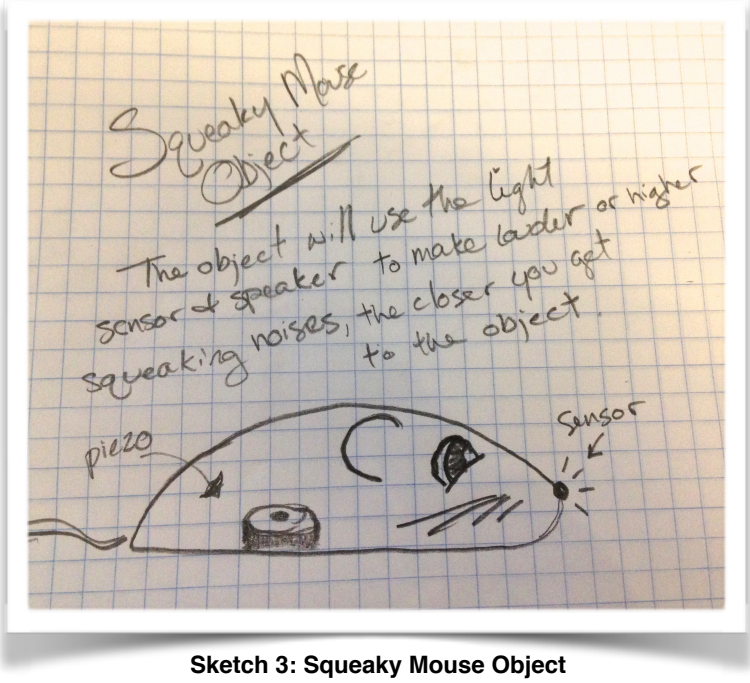
Prototypes

I had tried the photosensor in class, and later tested the piezo at home.

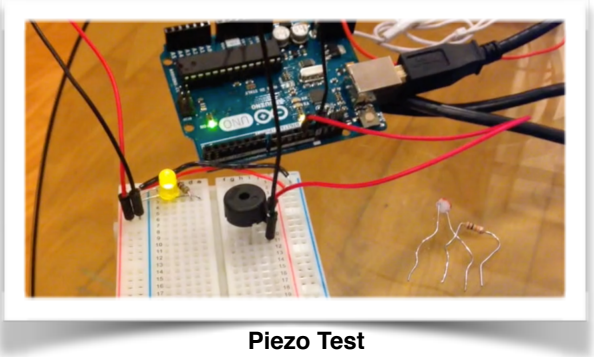
I decided to try make an object that would make noise (albeit grating, squeaky piezo noise), the closer the viewer got to the object. I decided in a mouse-like form for the object, and experimented with several arrangements of the input and output devices.



Photosensor Test
(see video online)



Sketch 3: Squeaky Mouse Object



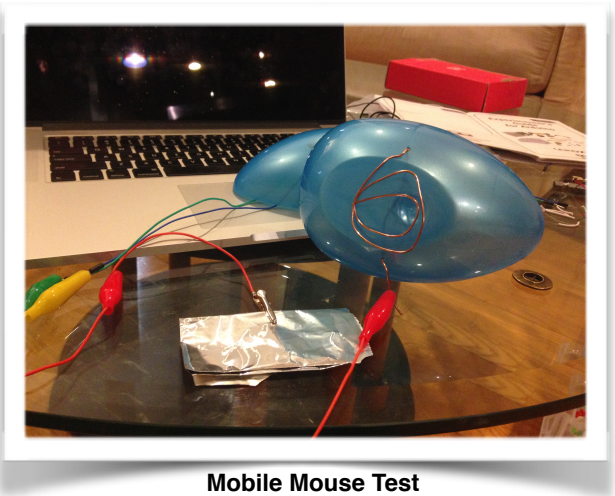
Piezo Test
(see video online)

Theremin Test
(see video online)

Sketch 3: Squeaky Mouse Object: Initial sketch of my final design. The photosensor is in nose of mouse, and piezo and Arduino are in mouse body.

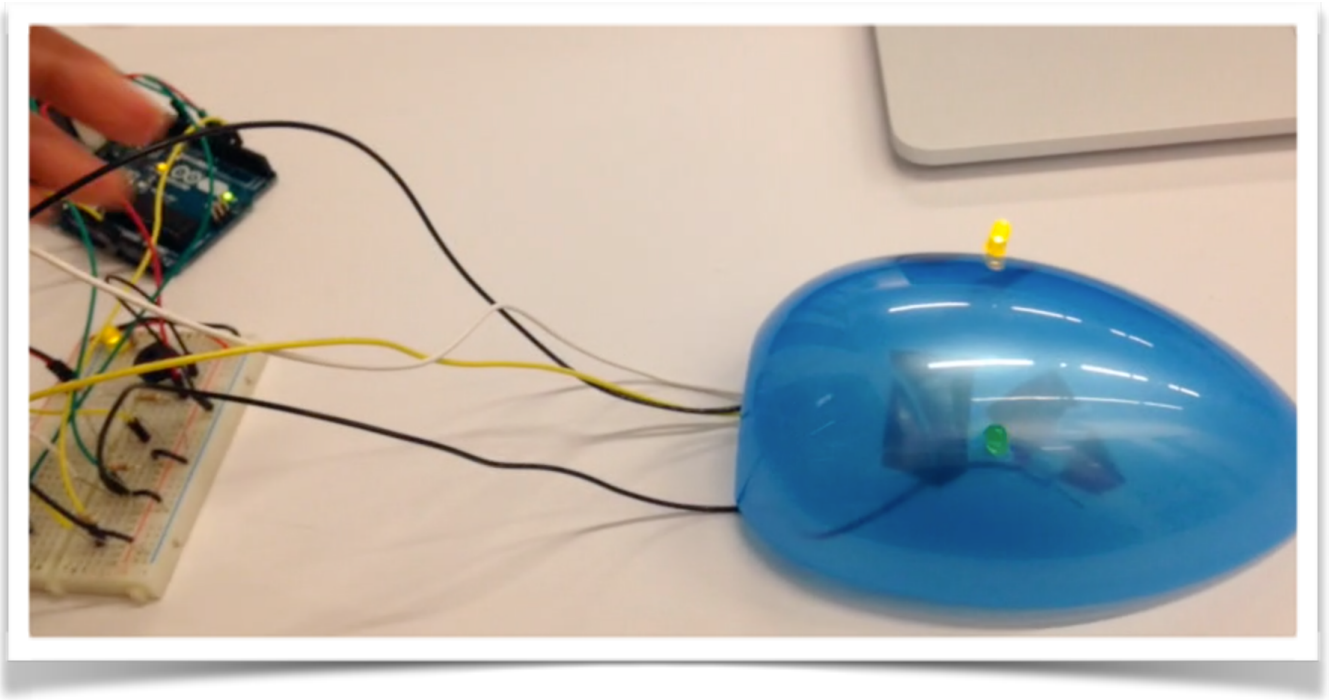
Theremin Test: Image from video of working photosensor and piezo inside of object. Unfortunately the connections in this set-up were very delicate and easily lost. (see video on blog)

Mobile Mouse Test: This design used a conductive surface or area over which the mouse could pass to complete the piezo circuit. This design allowed for too much movement, and I was unable to keep the circuits for my core functions (photosensor & piezo) in tact.



Mobile Mouse Test

Final Design



For the final design of Squeaky Mouse Object, I moved both the photosensor and piezo+breadboard 'offstage.' This design allowed for much more stable circuitry than in my earlier prototypes. The mouse's tail was preserved by it's connection to the external breadboard.

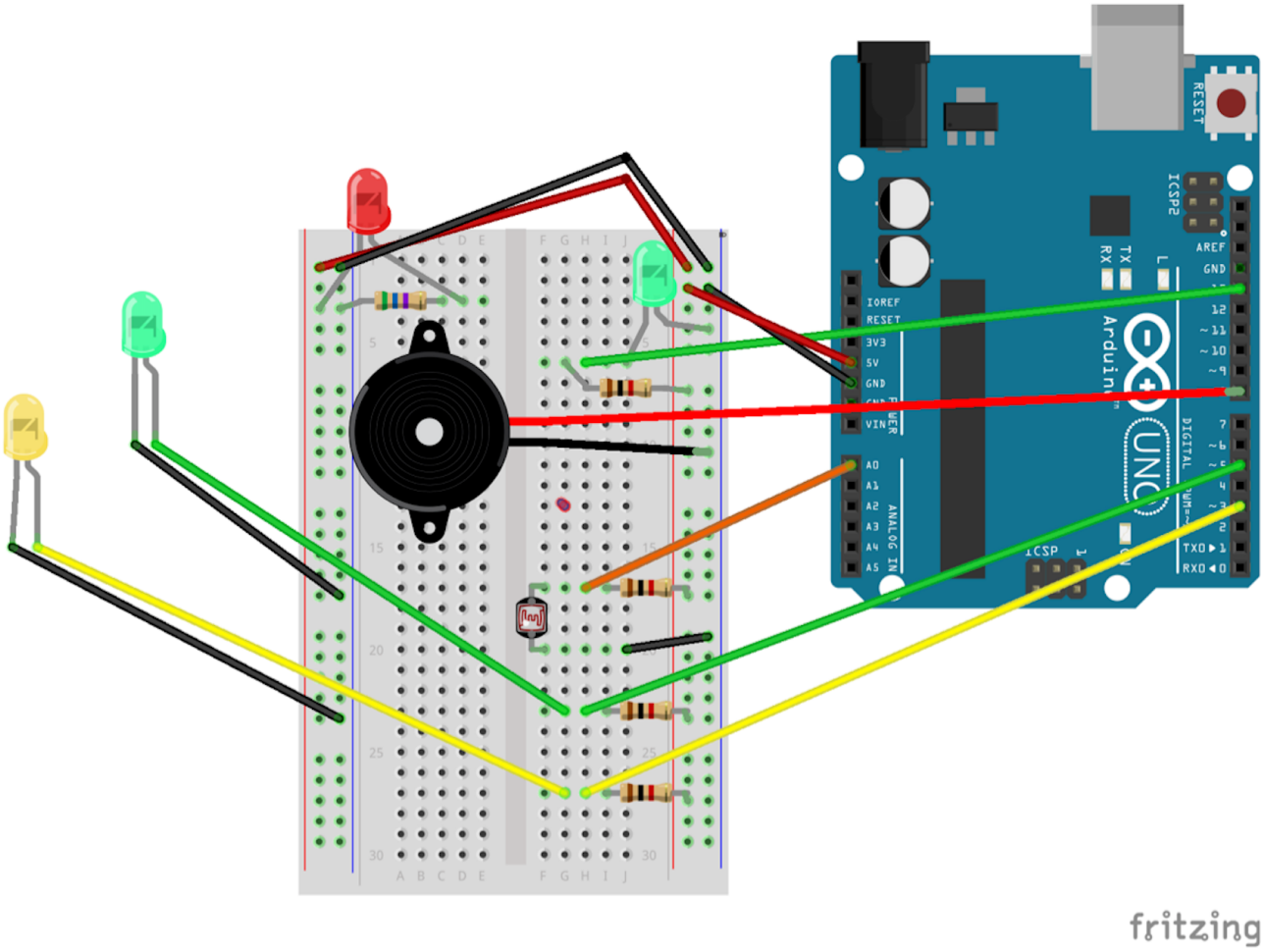
I/O & Switch
For this project, I used the light values read from the photosensor as my custom switch. The 'multi-level' or variable photosensor input was converted into two forms of output - analog (piezo) and digital (LED).

To use these values for piezo output, I learned map(), which re-maps one number range to another. In this project, it remapped my light input values, which tended to fall between 700 and 950, to a range that could be used by tone(). I used tone values of 100-4000Hz, however this was an arbitrary decision based on ranges used in other theremin projects I found on line. (Multiple online sources place the range of human hearing to be between 20Hz-20kHz, and state that sensitivity is best at the lower frequencies, generally below 5,000Hz.)

I added the LEDs to the mouse in order to experiment with a digital output. To convert the analog light readings to a digital switch, I took the initial input range gathered during the photosensor calibration period (see code), and calculated a threshold for 'high' and 'low' values based on the midpoint of this range. If the current sensor reading fell below the threshold (lower tones), the green LED would turn on; if readings were above the threshold (higher tones), the yellow LED would turn on.

I intended my output to represent the mouse's state of alertness. The yellow light and higher tones meant that my hand (potential danger) was nearer to the mouse.

Circuit Schematic



Pseudocode

```
//////////////////////////////////////////////
/* HELEN"S PSEUDO CODE

*** SET-UP ***
Set up & establish arduino connection with all components
    - mouse eye 1, green LED, OUTPUT for low tones
    - mouse eye 2, yellow LED, OUTPUT for high tones
    - calibrator, LED, OUTPUT will display light during millis calibration count
    - photosensor, analog INPUT, reads light
    - piezo, digital OUPUT, sound

Set up variables to hold high and low
(I found this brilliant calibration method in Scott's code. Previously, I had attempted to
hard code sensorHigh and sensorLow based on values found in serial printouts.)

   initialize sensorValue, which is a variable for storing the current photosensor value

   initialize sensorLow
      SensorLow is set to a value near the upper limit of the photosensor serial reads. By
      purposely starting the variable at a high value we can be sure to overwrite it with an
      appropriate low value during the calibration loop.
      After calibration, this variable will hold the lower light limit for our map function.

    initialize sensorHigh to 0
      By starting at 0 we can be sure to overwrite it with an appropriate high value during
      the calibration loop.
      After calibration, this variable will hold the upper light limit for our map function.

   initialize eyeDiff, which is a variable that stores the difference b/t sensorHigh &
   sensorLow
      eyeDiff/2 will be used to as a switch b/t the mouse's green (low tone range) and yellow
      (high tone range) eyes

Run calibration for 5s using mills()
  reset sensorHigh value
     Each time a value is recorded that is higher than the current state of sensorHigh,
     rewrite sensorHigh with the new high value.

  reset sensorLow value
      Each time a value is recorded that is lower than the current state of sensorLow,
      rewrite sensorLow with the new low value.

  turn off calibrator LED to signal the end of the mills calibration period

  calculate eyeDiff
      (sensorHigh-sensorLow)/2

Check the current values of sensorHigh, sensorLow, & eyeDiff via Serial.println()


*** LOOP ***
Read the input from the photosensor and assign it to sensorValue

Output light based on sensorValue data
- green mouse eye will light when low tones are read
- yellow mouse eye will light when high tones are read

Translate the sensorValue data to tones values using map()
- store the mapped tone values in a new variable called pitch

Output sounds to piezo based on pitch values


END PSEUDO CODE */
//////////////////////////////////////////////
```

Arduino Code

https://github.com/helencarey/StupidPetTricks_I/commit/a45d2b204a2a12b51c043fc9b607996b9371d606

References that I would like to explore in future:

Understanding Analog Output - Tone (Video Tutorial)
https://vimeo.com/93610177

Arduino Tutorial: Play a Melody using the tone() function
http://arduino.cc/en/Tutorial/Tone

Arduino Tutorial: How to play an Melody on a Piezo
http://www.arduino.cc/en/Tutorial/PlayMelody

Library of songs written for piezo and arduio tone()
http://www.princetronics.com/supermariothemesong/

Arduino Tutorial: How to play a with a Speaker
http://playground.arduino.cc/Code/MusicalAlgoFun

Arduino Tone Library Documentation
https://code.google.com/p/rogue-code/wiki/ToneLibraryDocumentation