```
/************************************************
  ProtoStax Capacitive Touch Piano Demo
  This is a example sketch for a Touch Piano using Arduino,
  Adafruit 12 x Capacitive Touch Shield for Arduino - MPR121 --> https://www.adafruit.com/product/2024,
  Piezo Buzzer PS1240,  Copper Foil Tape with Conductive Adhesive,  and
  ProtoStax for Arduino --> https://www.protostax.com/products/protostax-for-arduino
  It uses conductive copper foil tape stuck on the top of a ProtoStax for Arduino enclosure to provide a
  one-octave piano keyboard, using the capacitive touch shield to sense touches, and playing the appropriate
  notes on a piezo buzzer.
  The following pins are used in this example -
  SCL and SDA pins for I2C communications with the MPR121, and pin 12 for the buzzer. This uses the tone
  library, which will also affect pins 3 and 11 (the tone library uses Timer2 which will conflict with these pins).
  You can also use the replacement NewTone library (tone --> NewTone, and noTone --> noNewTone), which is smaller,
  has faster execution and uses Timer1 which can resolve conflicts with pins 3 and 11
  (but which conflicts with pins 9 and 10 instead)
  This takes some pointers from the MPR121test example sketch by Adafruit/Limor Fried.
  Written by Sridhar Rajagopal for ProtoStax
  BSD license. All text above must be included in any redistribution
 */
#include <Wire.h>
#include <JC_Button.h>
#include "Adafruit_MPR121.h"
#include "pitches.h"
#include "noteDurations.h"
#ifndef _BV
#define _BV(bit) (1 << (bit))
#endif
```

```
#define  SW1     2        //Connect a tactile button switch (or something similar)
                    //from Arduino pin 2 to ground.
#define  SW2     3        //Connect a tactile button switch (or something similar)
                    //from Arduino pin 3 to ground.
#define PULLUP true       //To keep things simple, we use the Arduino's internal pullup resistor.
#define INVERT true       //Since the pullup resistor will keep the pin high unless the
                    //switch is closed, this is negative logic, i.e. a high state
                    //means the button is NOT pressed. (Assuming a normally open switch.)
#define DEBOUNCE_MS 20     //A debounce time of 20 milliseconds usually works well for tactile button switches.
#define LONG_PRESS 1000    //We define a "long press" to be 1000 milliseconds.
Button upBtn(SW1, PULLUP, INVERT, DEBOUNCE_MS);    //Declare the button
Button downBtn(SW2, PULLUP, INVERT, DEBOUNCE_MS);    //Declare the button
// pitches.h defines all the frequencies for the different notes in different octaves
// Let us pick one octave for our keyboard below, so it is easy to change it in one place
// For example, if you want to go down an octave, change NOTE_C7 to NOTE_C6 and so on
#define NOTE_C  NOTE_C7
#define NOTE_CS NOTE_CS7
#define NOTE_D  NOTE_D7
#define NOTE_DS NOTE_DS7
#define NOTE_E  NOTE_E7
#define NOTE_F  NOTE_F7
#define NOTE_FS NOTE_FS7
#define NOTE_G  NOTE_G7
#define NOTE_GS NOTE_GS7
#define NOTE_A  NOTE_A7
#define NOTE_AS NOTE_AS7
#define NOTE_B  NOTE_B7
// These correspond to the 12 physical capacitive touch keys of the Piano
```

```
// Feel free to change the mapping to your own!

// notes in the scale:

int scale[7][12] = {

  { NOTE_C1, NOTE_CS1, NOTE_D1, NOTE_DS1, NOTE_E1, NOTE_F1, NOTE_FS1, NOTE_G1, NOTE_GS1, NOTE_A1, NOTE_AS1, NOTE_B1 },

  { NOTE_C2, NOTE_CS2, NOTE_D2, NOTE_DS2, NOTE_E2, NOTE_F2, NOTE_FS2, NOTE_G2, NOTE_GS2, NOTE_A2, NOTE_AS2, NOTE_B2 },

  { NOTE_C3, NOTE_CS3, NOTE_D3, NOTE_DS3, NOTE_E3, NOTE_F3, NOTE_FS3, NOTE_G3, NOTE_GS3, NOTE_A3, NOTE_AS3, NOTE_B3 },

  { NOTE_C4, NOTE_CS4, NOTE_D4, NOTE_DS4, NOTE_E4, NOTE_F4, NOTE_FS4, NOTE_G4, NOTE_GS4, NOTE_A4, NOTE_AS4, NOTE_B4 },

  { NOTE_C5, NOTE_CS5, NOTE_D5, NOTE_DS5, NOTE_E5, NOTE_F5, NOTE_FS5, NOTE_G5, NOTE_GS5, NOTE_A5, NOTE_AS5, NOTE_B5 },

  { NOTE_C6, NOTE_CS6, NOTE_D6, NOTE_DS6, NOTE_E6, NOTE_F6, NOTE_FS6, NOTE_G6, NOTE_GS6, NOTE_A6, NOTE_AS6, NOTE_B6 },

  { NOTE_C7, NOTE_CS7, NOTE_D7, NOTE_DS7, NOTE_E7, NOTE_F7, NOTE_FS7, NOTE_G7, NOTE_GS7, NOTE_A7, NOTE_AS7, NOTE_B7 },

};

int octaveNum = 4;

// The Piezo uses pin 12 in our example. Change as needed. Remember that you cannot use pins 3 and 11

// when using the tone library

#define TONE_PIN 12

// All of our tunes have the same characteristics

// A Note in a melody has a frequency and a duration (1 = whole note, 2 = half note, etc)

// A melody consists of an array of such Notes (the melody)

// playTunes takes a pointer to the melody array of Notes, as well as a speedup factor.

// The speedup factor is to facilitate playing at a higher speed (1 plays at intended speed,

// 2 at twice the speed and so on) without having to modify the noteDurations array

typedef struct Note {

  int frequency;

  float duration;
```

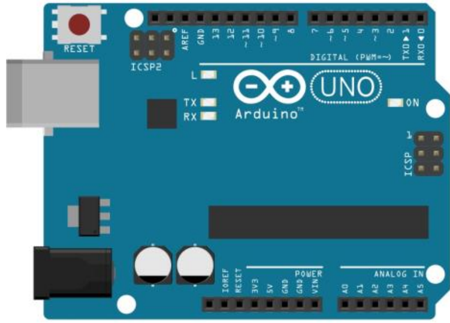
```
} Note;

// This melody is Twinkle Twinkle Little Star!

Note melody[] = {

  {NOTE_C, NOTE_HALF}, {NOTE_C, NOTE_HALF}, {NOTE_G, NOTE_HALF}, {NOTE_G, NOTE_HALF}, {NOTE_A, NOTE_HALF}, {NOTE_A, NOTE_HALF}, {NOTE_G, NOTE_WHOLE},

  {NOTE_F, NOTE_HALF}, {NOTE_F, NOTE_HALF}, {NOTE_E, NOTE_HALF}, {NOTE_E, NOTE_HALF}, {NOTE_D, NOTE_HALF}, {NOTE_D, NOTE_HALF}, {NOTE_C, NOTE_WHOLE},

  {NOTE_G, NOTE_HALF}, {NOTE_G, NOTE_HALF}, {NOTE_F, NOTE_HALF}, {NOTE_F, NOTE_HALF}, {NOTE_E, NOTE_HALF}, {NOTE_E, NOTE_HALF}, {NOTE_D, NOTE_WHOLE},

  {NOTE_G, NOTE_HALF}, {NOTE_G, NOTE_HALF}, {NOTE_F, NOTE_HALF}, {NOTE_F, NOTE_HALF}, {NOTE_E, NOTE_HALF}, {NOTE_E, NOTE_HALF}, {NOTE_D, NOTE_WHOLE},

  {NOTE_C, NOTE_HALF}, {NOTE_C, NOTE_HALF}, {NOTE_G, NOTE_HALF}, {NOTE_G, NOTE_HALF}, {NOTE_A, NOTE_HALF}, {NOTE_A, NOTE_HALF}, {NOTE_G, NOTE_WHOLE},

  {NOTE_F, NOTE_HALF}, {NOTE_F, NOTE_HALF}, {NOTE_E, NOTE_HALF}, {NOTE_E, NOTE_HALF}, {NOTE_D, NOTE_HALF}, {NOTE_D, NOTE_HALF}, {NOTE_C, NOTE_WHOLE},

};

// This melody is playing all the notes of the octave from C up to B and back down to C

Note melody2[] = {

  {NOTE_C, NOTE_WHOLE}, {NOTE_CS, NOTE_WHOLE}, {NOTE_D, NOTE_WHOLE}, {NOTE_DS, NOTE_WHOLE}, {NOTE_E, NOTE_WHOLE}, {NOTE_F, NOTE_WHOLE}, {NOTE_FS, NOTE_WHOLE}, {NOTE_G, NOTE_WHOLE}, {NOTE_GS, NOTE_WHOLE}, {NOTE_A, NOTE_WHOLE}, {NOTE_AS, NOTE_WHOLE}, {NOTE_B, NOTE_WHOLE},

  {NOTE_AS, NOTE_WHOLE}, {NOTE_A, NOTE_WHOLE}, {NOTE_GS, NOTE_WHOLE}, {NOTE_G, NOTE_WHOLE}, {NOTE_FS, NOTE_WHOLE}, {NOTE_F, NOTE_WHOLE}, {NOTE_E, NOTE_WHOLE}, {NOTE_DS, NOTE_WHOLE}, {NOTE_D, NOTE_WHOLE}, {NOTE_CS, NOTE_WHOLE}, {NOTE_C, NOTE_WHOLE}

};

// Oh Susanna!

Note melody3[] = {

  {NOTE_C, NOTE_EIGHTH}, {NOTE_D, NOTE_EIGHTH}, {NOTE_E, NOTE_QUARTER}, {NOTE_G, NOTE_QUARTER}, {NOTE_G, NOTE_QUARTER}, {NOTE_A, NOTE_QUARTER}, {NOTE_G, NOTE_QUARTER}, {NOTE_E, NOTE_QUARTER}, {NOTE_C, DOTTED(NOTE_QUARTER)}, {NOTE_D, NOTE_EIGHTH}, {NOTE_E, NOTE_QUARTER},  {NOTE_E, NOTE_QUARTER}, {NOTE_D, NOTE_QUARTER}, {NOTE_C, NOTE_QUARTER}, {NOTE_D, DOTTED(NOTE_HALF)},

  {NOTE_C, NOTE_EIGHTH}, {NOTE_D, NOTE_EIGHTH}, {NOTE_E, NOTE_QUARTER}, {NOTE_G, NOTE_QUARTER}, {NOTE_G, DOTTED(NOTE_QUARTER)}, {NOTE_A, NOTE_EIGHTH}, {NOTE_G, NOTE_QUARTER}, {NOTE_E, NOTE_QUARTER}, {NOTE_C, DOTTED(NOTE_QUARTER)}, {NOTE_D, NOTE_EIGHTH}, {NOTE_E, NOTE_QUARTER}, {NOTE_E, NOTE_QUARTER}, {NOTE_D, NOTE_QUARTER}, {NOTE_D, NOTE_QUARTER}, {NOTE_C, DOTTED(NOTE_HALF)},
```

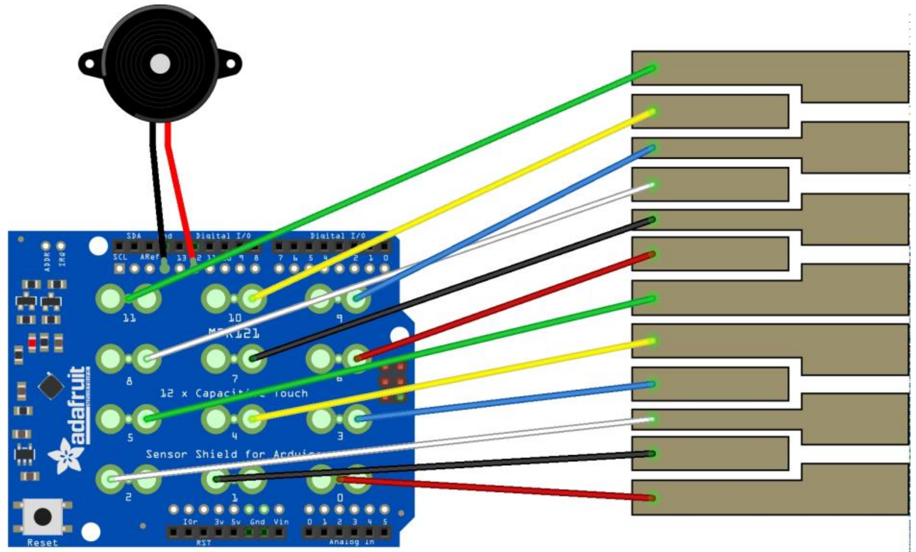
```
  {NOTE_C, NOTE_EIGHTH}, {NOTE_D, NOTE_EIGHTH}, {NOTE_E, NOTE_QUARTER}, {NOTE_G, NOTE_QUARTER},
{NOTE_G, NOTE_QUARTER}, {NOTE_A, NOTE_QUARTER}, {NOTE_G, NOTE_QUARTER}, {NOTE_E, NOTE_QUARTER},
{NOTE_C, DOTTED(NOTE_QUARTER)}, {NOTE_D, NOTE_EIGHTH}, {NOTE_E, NOTE_QUARTER},  {NOTE_E,
NOTE_QUARTER}, {NOTE_D, NOTE_QUARTER}, {NOTE_C, NOTE_QUARTER}, {NOTE_D, DOTTED(NOTE_HALF)},

  {NOTE_C, NOTE_EIGHTH}, {NOTE_D, NOTE_EIGHTH}, {NOTE_E, NOTE_QUARTER}, {NOTE_G, NOTE_QUARTER},
{NOTE_G, DOTTED(NOTE_QUARTER)}, {NOTE_A, NOTE_EIGHTH}, {NOTE_G, NOTE_QUARTER}, {NOTE_E,
NOTE_QUARTER}, {NOTE_C, DOTTED(NOTE_QUARTER)}, {NOTE_D, NOTE_EIGHTH}, {NOTE_E, NOTE_QUARTER},
{NOTE_E, NOTE_QUARTER}, {NOTE_D, NOTE_QUARTER}, {NOTE_D, NOTE_QUARTER}, {NOTE_C, NOTE_WHOLE},

  {NOTE_F, NOTE_HALF}, {NOTE_F, NOTE_HALF}, {NOTE_A, NOTE_QUARTER}, {NOTE_A, NOTE_HALF}, {NOTE_A,
NOTE_QUARTER}, {NOTE_G, NOTE_QUARTER}, {NOTE_G, NOTE_QUARTER}, {NOTE_E, NOTE_QUARTER}, {NOTE_C,
NOTE_QUARTER}, {NOTE_D, DOTTED(NOTE_HALF)},

  {NOTE_C, NOTE_EIGHTH}, {NOTE_D, NOTE_EIGHTH}, {NOTE_E, NOTE_QUARTER}, {NOTE_G, NOTE_QUARTER},
{NOTE_G, DOTTED(NOTE_QUARTER)}, {NOTE_A, NOTE_EIGHTH}, {NOTE_G, NOTE_QUARTER}, {NOTE_E,
NOTE_QUARTER}, {NOTE_C, DOTTED(NOTE_QUARTER)}, {NOTE_D, NOTE_EIGHTH}, {NOTE_E, NOTE_QUARTER},
{NOTE_E, NOTE_QUARTER}, {NOTE_D, NOTE_QUARTER}, {NOTE_D, NOTE_QUARTER}, {NOTE_C, NOTE_WHOLE},

};

// Frere Jacques / Brother John

Note melody4[] = {

  {NOTE_C, NOTE_QUARTER}, {NOTE_D, NOTE_QUARTER}, {NOTE_E, NOTE_QUARTER}, {NOTE_C, NOTE_QUARTER},
{NOTE_C, NOTE_QUARTER}, {NOTE_D, NOTE_QUARTER}, {NOTE_E, NOTE_QUARTER}, {NOTE_C, NOTE_QUARTER},

  {NOTE_E, NOTE_QUARTER}, {NOTE_F, NOTE_QUARTER}, {NOTE_G, NOTE_HALF}, {NOTE_E, NOTE_QUARTER}, {NOTE_F,
NOTE_QUARTER}, {NOTE_G, NOTE_HALF},

  {NOTE_G, NOTE_EIGHTH}, {NOTE_A, NOTE_EIGHTH}, {NOTE_G, NOTE_EIGHTH}, {NOTE_F, NOTE_EIGHTH}, {NOTE_E,
NOTE_QUARTER}, {NOTE_C, NOTE_QUARTER}, {NOTE_G, NOTE_EIGHTH}, {NOTE_A, NOTE_EIGHTH}, {NOTE_G,
NOTE_EIGHTH}, {NOTE_F, NOTE_EIGHTH}, {NOTE_E, NOTE_QUARTER}, {NOTE_C, NOTE_QUARTER},

  {NOTE_D, NOTE_QUARTER}, {NOTE_C, NOTE_QUARTER}, {NOTE_C, NOTE_HALF}, {NOTE_D, NOTE_QUARTER},
{NOTE_C, NOTE_QUARTER}, {NOTE_C, NOTE_HALF},

};

#define MELODY_LENGTH(m) (sizeof(m)/sizeof(m[0]))

Adafruit_MPR121 cap = Adafruit_MPR121();

// Keeps track of the last pins touched

// so we know when buttons are 'released'

uint16_t lasttouched = 0;

uint16_t currtouched = 0;
```

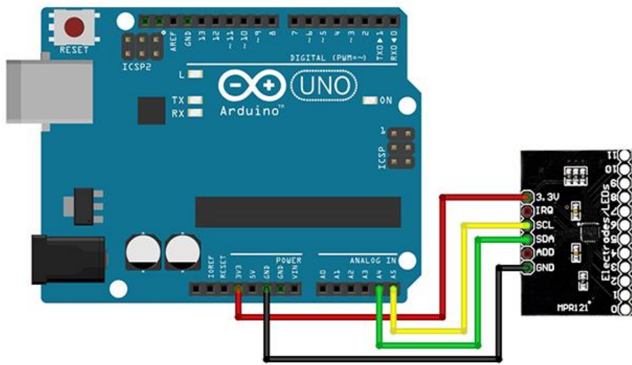
```
void setup() {
  Serial.begin(9600);
  while (!Serial) { // needed to keep leonardo/micro from starting too fast!
    delay(10);
  }
  Serial.println(F("ProtoStax Piano Demo"));
  // Default address is 0x5A, if tied to 3.3V its 0x5B
  // If tied to SDA its 0x5C and if SCL then 0x5D
  if (!cap.begin(0x5A)) {
    Serial.println(F("MPR121 not found, check wiring?"));
    while (1);
  }
  Serial.println(F("MPR121 found!"));
  playTune(melody2, 23, 48);
}
void loop() {
  upBtn.read();
  downBtn.read();
  static int lastOctave = 4;
  lastOctave = octaveNum;
  if (upBtn.isPressed()) {
    octaveNum = 5;
  } else if (downBtn.isPressed()) {
    octaveNum = 3;
  } else {
    octaveNum = 4;
  }
  // Get the currently touched pads
```

```
  currtouched = cap.touched();

  // Play tunes if our 'secret' combination of keys is pressed!

  if (currtouched == 2193) { // C E G B

    playTune(melody, MELODY_LENGTH(melody), 2); // Play Twinkle Twinkle Little Star

  } else if (currtouched == 181) { // C D E F G

    playTune(melody2, MELODY_LENGTH(melody2), 48); // Play scale slide up and down

  } else if (currtouched == 2565) { // C D A B

    playTune(melody3, MELODY_LENGTH(melody3), 1); // Play Oh Susanna!

  } else if (currtouched == 3075) { // C C# A# B

    playTune(melody4, MELODY_LENGTH(melody4), 1);  // Play Frere Jacques

  } else {

    for (uint8_t i=0; i<12; i++) {

      // if it *is* touched and *wasnt* touched before, play the corresponding tone!

      if ((currtouched & _BV(i)) && (!(lasttouched & _BV(i)) || (lastOctave != octaveNum) )) {

        // Serial.print("currtouched = "); Serial.print(currtouched); Serial.print(" lasttouched = "); Serial.print(lasttouched);
Serial.print(" "); Serial.print(i); Serial.println(" touched");

        tone(TONE_PIN, scale[octaveNum][i]);

      }

      // if it *was* touched and now *isnt*, and additionally, no button is being pressed, then stop the tone

      // This allows for smooth transitions between notes as you slide you finger over the keys

      // This also allows for sustain of a given note, by touching any key and keeping it touched - you can press another key
and it will play that key, but won't stop playing it

      // until you let go of your first key - kind of like the sustain pedal of the piano!

      if (!(currtouched & _BV(i)) && (lasttouched & _BV(i)) && !currtouched) {

      //if (!(currtouched & _BV(i)) && (lasttouched & _BV(i))) {

        // Serial.print("currtouched = "); Serial.print(currtouched); Serial.print(" lasttouched = "); Serial.print(lasttouched);
Serial.print(" "); Serial.print(i); Serial.println(" released");

        noTone(TONE_PIN);

      }
```

```
    }
  }
  // reset our state
  lasttouched = currtouched;
}
// This function is used to play a given melody
// Arguments are:
// array of Notes for the melody
// size of array
// speedup factor - higher number plays the melody faster
void playTune(Note *m, int mSize, int speedUp) {
  noTone(TONE_PIN); // Start with a clean slate
  for (int thisNote = 0; thisNote < mSize; thisNote++) {
    // to calculate the note duration, take one second multiplies by the note type.
    //e.g. quarter note = 1000.0 * 0.25, eighth note = 1000 * 1/8 (0.125), etc.
    // reduce the duration by the speedup factor to increase the speed of playing
    // by an appropriate amount
    int noteDuration = 1000.0 * m[thisNote].duration / speedUp;
    tone(TONE_PIN, m[thisNote].frequency, noteDuration);
    // to distinguish the notes, set a minimum time between them.
    // the note's duration + 30% seems to work well:
    int pauseBetweenNotes = noteDuration * 1.30;
    delay(pauseBetweenNotes);
    // stop the tone playing:
    noTone(TONE_PIN);
  }
  delay(100);
}
```


RESET
ICSP2
UNO
Arduino
DIGITAL (PWM=~)
ON
TX
RX
ICSP
POWER
ANALOG IN


Digital I/O
MPR121
12 x Capacitive Touch
Sensor Shield for Arduino
adafruit
Reset
fritzing


RESET
ICSP2
UNO
Arduino
DIGITAL (PWM=~)
ON
TX
RX
ICSP
POWER
ANALOG IN
3.3V
IRQ
SCL
SDA
ADD
GND
Electrodes/LEDs
MPR121

www.Electropeak.com
BY SA