// Pin definitions

const int knockSensor = 0; // Piezo sensor on pin 0.

const int programSwitch = 2; // If this is high we program a new code.

const int lockMotor = 3; // Gear motor used to turn the lock.

const int redLED = 4; // Status LED

const int greenLED = 5; // Status LED

// Tuning constants. Could be made vars and hoooked to potentiometers for soft configuration, etc.

const int threshold = 5; // Minimum signal from the piezo to register as a knock

const int rejectValue = 25; // If an individual knock is off by this percentage of a knock we don't unlock..

const int averageRejectValue = 15; // If the average timing of the knocks is off by this percent we don't unlock.

const int knockFadeTime = 150; // milliseconds we allow a knock to fade before we listen for another one. (Debounce timer.)

const int lockTurnTime = 650; // milliseconds that we run the motor to get it to go a half turn.

const int maximumKnocks = 20; // Maximum number of knocks to listen for.

const int knockComplete = 1200; // Longest time to wait for a knock before we assume that it's finished.

// Variables.

int secretCode[maximumKnocks] = {100, 100, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}; // Initial setup: "Shave and a Hair Cut, two bits."

int knockReadings[maximumKnocks]; // When someone knocks this array fills with delays between knocks.

int knockSensorValue = 0; // Last reading of the knock sensor.

int programButtonPressed = false; // Flag so we remember the programming button setting at the end of the cycle.

void setup() {

pinMode(lockMotor, OUTPUT);

pinMode(redLED, OUTPUT);

pinMode(greenLED, OUTPUT);

pinMode(programSwitch, INPUT);

Serial.begin(9600); // Uncomment the Serial.bla lines for debugging.

Serial.println("Program start."); // but feel free to comment them out after it's working right.

digitalWrite(greenLED, HIGH); // Green LED on, everything is go.

}

void loop() {

// Listen for any knock at all.

knockSensorValue = analogRead(knockSensor);

if (knockSensorValue >=threshold){

listenToSecretKnock();

}

}

// Records the timing of knocks.

void listenToSecretKnock(){

Serial.println("knock starting");

int i = 0;

// First lets reset the listening array.

for (i=0;i<maximumKnocks;i++){

knockReadings[i]=0;

}

int currentKnockNumber=0; // Incrementer for the array.

int startTime=millis(); // Reference for when this knock started.

int now=millis();

digitalWrite(greenLED, LOW); // we blink the LED for a bit as a visual indicator of the knock.

{

digitalWrite(redLED, LOW); // and the red one too if we're programming a new knock.

}

delay(knockFadeTime); // wait for this peak to fade before we listen to the next one.

digitalWrite(greenLED, HIGH);

do {

//listen for the next knock or wait for it to timeout.

knockSensorValue = analogRead(knockSensor);

if (knockSensorValue >=threshold){ //got another knock...

//record the delay time.

Serial.println("knock.");

now=millis();

knockReadings[currentKnockNumber] = now-startTime;

currentKnockNumber ++; //increment the counter

startTime=now;

// and reset our timer for the next knock

digitalWrite(greenLED, LOW);

delay(knockFadeTime); // again, a little delay to let the knock decay.

digitalWrite(greenLED, HIGH);

}

now=millis();

//did we timeout or run out of knocks?

} while ((now-startTime < knockComplete) && (currentKnockNumber < maximumKnocks));

if (validateKnock() == true){

triggerDoorUnlock();

}else{

Serial.println("Secret knock failed.");

digitalWrite(greenLED, LOW); // We didn't unlock, so blink the red LED as visual feedback.

for (int i=0;i<4;i++){

digitalWrite(redLED, HIGH);

delay(100);

digitalWrite(redLED, LOW);

delay(100);

}

digitalWrite(greenLED, HIGH);

}

//we've got our knock recorded, lets see if it's valid

}

// Runs the motor (or whatever) to unlock the door.

void triggerDoorUnlock(){

Serial.println("Door unlocked!");

int i=0;

// turn the motor on for a bit.

digitalWrite(lockMotor, HIGH);

digitalWrite(greenLED, HIGH); // And the green LED too.

delay (lockTurnTime); // Wait a bit.

digitalWrite(lockMotor, LOW); // Turn the motor off.

// Blink the green LED a few times for more visual feedback.

for (i=0; i < 5; i++){

digitalWrite(greenLED, LOW);

delay(100);

digitalWrite(greenLED, HIGH);

delay(100);

}

}

// Sees if our knock matches the secret.

// returns true if it's a good knock, false if it's not.

// todo: break it into smaller functions for readability.

boolean validateKnock(){

int i=0;

// simplest check first: Did we get the right number of knocks?

int currentKnockCount = 0;

int secretKnockCount = 0;

int maxKnockInterval = 0; // We use this later to normalize the times.

for (i=0;i<maximumKnocks;i++){

if (knockReadings[i] > 0){

currentKnockCount++;

}

if (secretCode[i] > 0){ //todo: precalculate this.

secretKnockCount++;

}

if (knockReadings[i] > maxKnockInterval){ // collect normalization data while we're looping.

maxKnockInterval = knockReadings[i];

}

}

/\* Now we compare the relative intervals of our knocks, not the absolute time between them.

(ie: if you do the same pattern slow or fast it should still open the door.)

This makes it less picky, which while making it less secure can also make it

less of a pain to use if you're tempo is a little slow or fast.

\*/

int totaltimeDifferences=0;

int timeDiff=0;

for (i=0;i<maximumKnocks;i++){ // Normalize the times

knockReadings[i]= map(knockReadings[i],0, maxKnockInterval, 0, 100);

timeDiff = abs(knockReadings[i]-secretCode[i]);

if (timeDiff > rejectValue){ // Individual value too far out of whack

return false;

}

totaltimeDifferences += timeDiff;

}

// It can also fail if the whole thing is too inaccurate.

if (totaltimeDifferences/secretKnockCount>averageRejectValue){

return false;

}

return true;

}