Rachel Choe & Rashida Doctor
Phys 124L
Fall 2018

"Smart Glove" Project Proposal-Report

Motivation and Overview

This project was motivated by the children's hand game Indiana Jones. It is an upcycling of the simple game, with sound effects, tracking of the score, and a visual display of what it going on. Through this project, people playing the game can enjoy sounds that go along with the game as well as not have to rely on integrity to figure out if someone won.

Functional Definition

Our device will track and differentiate human hand movements in a hand game called Indiana Jones using flex strips and tilt sensors for each player.

The hand game instructions are:
Two players will try to win a game by "shooting" each other.

The three motions:
1) Shooting motion: Fold in the last three fingers and leave the pointer finger and thumb out to make the shape of a gun
2) Reload: Bend elbows and move hands up, leaving hands in shape of gun
3) Shield: Unfold all fingers and cross arms across chest

How those motions play out:
1)Shooting will "kill" your partner, and if your partner is not shielded or making a shooting motion also, then they will lose that game. If both players are making a gun shape, then we are at a stalemate and the game will go on.
2)Reloading will reload your gun; if you shoot your gun barrel will be empty, so you have to reload before you can shoot again. You can alternatively not shoot and collect your reloads; five reloads without shooting lets you shoot an "ultimate bullet" which can go through a shield.
3)Shielding will protect you from a bullet, except for the "ultimate bullet" stated above.

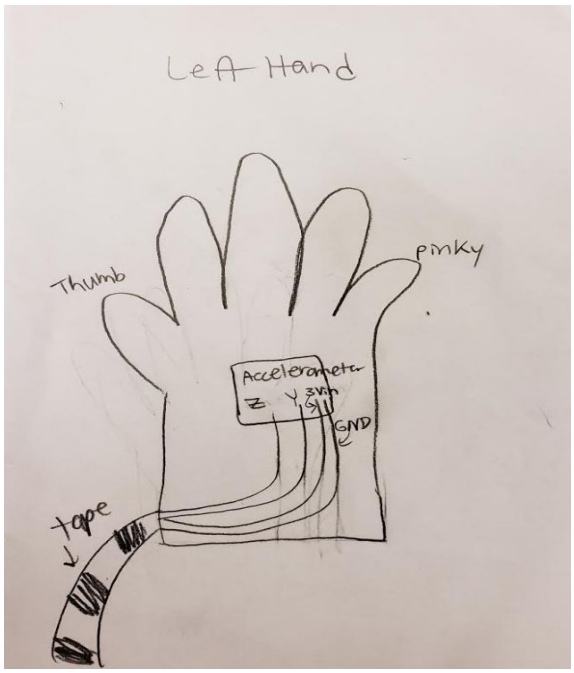
In between each motion the players must use unfold all fingers, and slap their thighs twice, in sync to the rhythm/ each other.

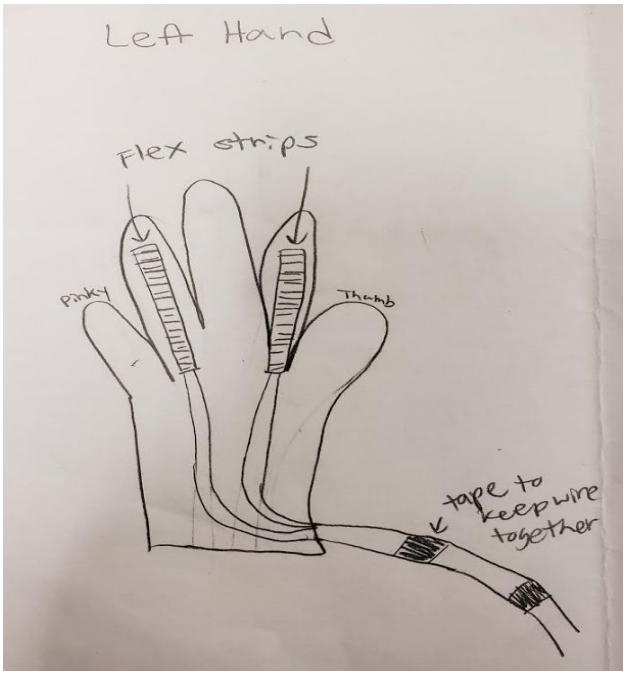
One flex strip and tilt sensor will go on a glove and sense if the hand or fingers have been bended or moved up or down.
The speaker needs to play audio at appropriate times, to keep the rhythm of the game and to add different sound effects for particular moves made and when a game is won.
The LCD will display the score between the two players and you can reset the score by pressing the reset button.

Sensors

Our sensing devices are the flex strips and accelerometers.
The flex strips are basically variable resistors. They will be used as part of a voltage divider and the input into the arduino will differ based on how much it is bent. We are using it to differentiate between a closed hand and an open hand.
The tilt sensor outputs values based on its orientation. By finding the ranges of outputs that correspond to the different motions, we can determine the position of the hand.

Mechanical Considerations

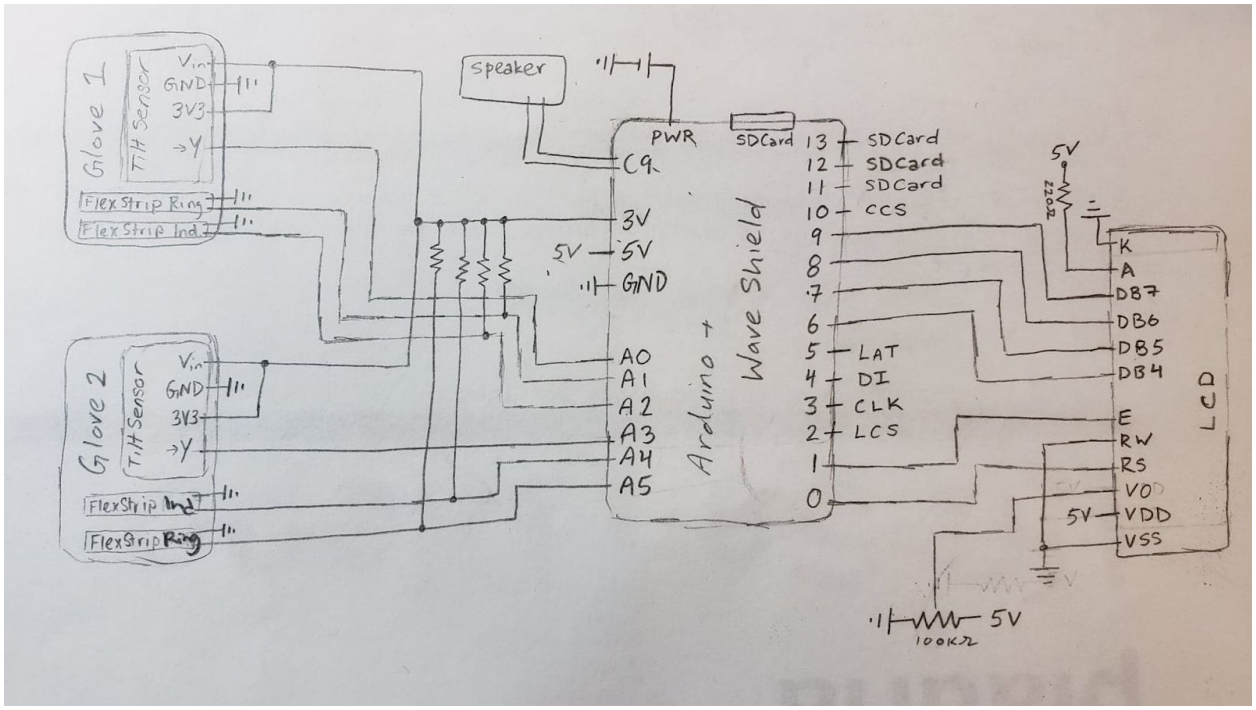




The Gloves: Each glove will hold the two sensors, a flex strip and a tilt sensor. There will be one glove per player, meant to be worn on their left hand. The flex strip will be on the inner side of the third finger of the glove and will be held on with either sewing thread or tape on two parts of the glove's finger, avoiding the middle joint for flexibility. The tilt sensors will be on the back, behind the palm and will be held on with strong tape wrapped around the body of the hand. These glove will have wires coming off of them, but those will be held together so that effectively a single cord will attach them to the rest of the set up. All wires will be taped up and the tilt sensors and flex strip components themselves will be wrapped in tape as safety precautions. We need to ensure that we use flexible wires for the ones spanning between the glove and the breadboard.
There will also be a speaker and LCD. The speaker will be attached to the wave shield with the SD card on it and the LCD will be wired on separately.

Electrical Considerations
We will build a circuit that uses a flex strips to differentiate between a folded finger and unfolded finger. The bending of the flex strip will cause resistance to change and we will use a voltage divider to detect the change.
Tilt sensors will be needed to measure the orientation.

We will also be using an LCD to display score, so it will take digital input from the arduino.
We will also utilize a speaker to keep the rhythm of the game going smoothly. The speaker will be set up so that it takes in the output from the SD card on the breadboard.
Power will come from the Arduino board, which will be connected to an external battery.



Interfaces
We will use one Arduino Uno, and the Serial monitor to track behavior and to debug. We will also be using an audio wave shield to play the speaker and as an sd card reader.

We will use all of the Uno's analog inputs (2 for each of the tilt sensors), and the other 2 analog inputs for the flex strips (1 for each strip).
We will use all digital output pins (6 for the LCD, 8 for the wave shield: 4 for the sd card, 4 for DAC).

Software
We will program in C, using Arduino IDE. We will use the standard library in addition to a library for using the wave shield and playing audio (WaveHC.h sourced from: Github).

We need to program it so that the code follows a certain rhythm. This will consist of a main loop with audio output, delays, sections of times with reading analog input, calculations, and then a output response in the form of audio and LCD display.
If there is at least one gun hand motion, it will play the shooting sound. If there are two shields, it will play the shield sound. If there is no gun, and there is a reload motion, it will play the reload sound. In between the motions, the speaker will play a tone sound twice to keep the rhythm of the game .

The trickiest part of this will probably be in calibrating the tilt sensor so that the measurements we get can be translated into something somewhat representative of what is happening in the analog world. We will use integration in code to measure the distance and directions of the tilt sensors, to decipher between up and horizontal movements of the glove.
We will also write code to decipher between analog levels for the unbended and bended flex strip.

Testing
We will test the unit as we are developing because it does not require any special conditions to work. While the final product will be battery powered so that it is mobile, while testing, we can use the computer input instead.

Safety
We will use insulating tape to wrap up all the wires that are attached to the glove that is close enough to the body to give harm. We will also wrap the tilt sensors and flex strips in insulating tape.

Parts and Reusability
Two pairs of Flex strips and tilt sensors(flex strips not available in lab, need to order).
The flex strips can be reused because they will be sewed onto the glove, so we would just need to cut the threads attaching them. We will purchase a set of gloves.
The tilt sensors will be taped to the glove, so they can also be reused.
We will use an arduino wave shield for the sd card reader to hook up the speaker, which is available in lab. Resistors to connect for the flex strips are available in lab.
The resistors, speaker, LCD, and SD card connector, and SD card can all be salvaged from this project and reused.

Expansion Options
One easy expansion is to have the arduino track the score for multiple games so that there can be multiple rounds.
If we have sufficient additional time, it would be fun to add the functionality to toggle between an additional game (Rock, Paper, Scissors). These games can make use of the same sensors to differentiate between those hand motions as well and it would use a very similar rhythm and scoring system.

De-scope Options
If the project ends up being more complicated than planned for, it can be made less complicated in a couple ways. One is to let go of the LCD display functionality since most of the information could be relayed via the speaker and the rest of it is not essential for the game. Another thing that can be descoped is that we can change the game from the original version to something that can be more easily differentiated by the sensors that we are using, perhaps taking out one

of the hand movements that is similar to another. We could also take out the sound effects, and keep the LCD instead so we can keep track of who won.

The Deliverables

The main deliverable is the game itself. Along with a demonstration of our game during finals week, we will write a report on how it works and include diagrams and other useful information. We will also create a video with an explanation of the device as well as a demonstration.

Project Difficulties

The LCD cause some difficulties because sometimes it wouldn't even show the stagnant letters we wanted to show on it. The backlight would always work, but sometimes it would update the score incorrectly with random symbols. We fixed this issue by rewiring our circuit so that the wires would be closer down to the boards and so it would be less disturbed by movements of the gloves/wires connecting gloves to the boards. Also the wire would often break off at soldered points, we used electrically insulating putty to keep the wires together better.

Code:

```
//Rachel & Rashida
//Fall 2018
//This program will determine what move is made:
//Gun, Shield, or Reload based on
//analog input of flex strip and accelerometers

// for the random number generator
# include <stdlib.h>

// Two games -- differentiate between them
int game;
const int IJ = 40;
const int RPS = 41;

// Indiana Jones moves
#define GUN 0
#define RELOAD 1
#define SHIELD 3

// Rock Paper Scissors moves
#define ROCK 0
#define PAPER 1
#define SCISSORS 3

//Player 1 Pins
const int plyr1TSPin= 0;
//copy over to rock paper scissors #define plyr1IndPin  1
const int plyr1RingPin=1;
const int plyr1IndPin=2;
//Player 2 pins
const int plyr2TSPin=3;
//copy over to rock paper scissors #define plyr2IndPin  4
const int plyr2RingPin=4;
const int plyr2IndPin=5;

//ACCELEROMETER
# define UP 0
# define DOWN 1
```

```
float TS_avg;
int Accel_Position(float analogin);

int TS_avg_1 = 0;
int TS_avg_2 = 0;


//FLEX STRIP
#define OPEN 1
#define CLOSED 0

int FS_ring_avg_1 = 0;
int FS_ring_avg_2 = 0;
int FS_ind_avg_1 = 0;
int FS_ind_avg_2 = 0;

//SOUND
// From Github for The Play6_HC Example

#include <FatReader.h>
#include <SdReader.h>
#include <avr/pgmspace.h>
#include "WaveUtil.h"
#include "WaveHC.h"
SdReader card;    // This object holds the information for the card
FatVolume vol;    // This holds the information for the partition on the card
FatReader root;   // This holds the information for the filesystem on the card
FatReader f;      // This holds the information for the file we're play
WaveHC wave;      // This is the only wave (audio) object, since we will only play one at
a time

void playcomplete(char *name);


// this handy function will return the number of bytes currently free in RAM, great for
debugging!
int freeRam(void)
{
  extern int  __bss_end;
```

```
  extern int  *__brkval;
  int free_memory;
  if((int)__brkval == 0) {
    free_memory = ((int)&free_memory) - ((int)&__bss_end);
  }
  else {
    free_memory = ((int)&free_memory) - ((int)__brkval);
  }
  return free_memory;
}

void sdErrorCheck(void)
{
  if (!card.errorCode()) return;
  putstring("\n\rSD I/O error: ");
  Serial.print(card.errorCode(), HEX);
  putstring(", ");
  Serial.println(card.errorData(), HEX);
  while(1);
}

//SCORING VARS
int store1 = 0;
int store2 = 0;
int WIN1 = 0;
int WIN2 = 0;
int RPSWIN1 = 0;
int RPSWIN2 = 0;


//LCD
#include <LiquidCrystal.h>
// initialize the library by associating any needed LCD interface pin
// with the arduino pin number it is connected to
const int rs=0, en= 1, d4=6, d5=7, d6=8, d7=9;
LiquidCrystal lcd(rs, en, d4, d5, d6, d7);


////////////////////////////////////////////////
```

```
//////////SETUP////////////////////////////////
/////////////////////////////////////////////////
void setup() {
 //Serial.begin(9600);

 // randon number generator
  randomSeed(analogRead(1));

 //set mode for accelerometer pins
  pinMode(A0,INPUT);
  pinMode(A1,INPUT);

  //setup for SOUND
   putstring_nl("WaveHC with 6 buttons");

   putstring("Free RAM: ");       // This can help with debugging, running out of RAM is
bad
 // Serial.println(freeRam());      // if this is under 150 bytes it may spell trouble!

  // Set the output pins for the DAC control. This pins are defined in the library
  pinMode(2, OUTPUT);
  pinMode(3, OUTPUT);
  pinMode(4, OUTPUT);
  pinMode(5, OUTPUT);

  // pin13 LED
  pinMode(13, OUTPUT);


  //  if (!card.init(true)) { //play with 4 MHz spi if 8MHz isn't working for you
  if (!card.init()) {         //play with 8 MHz spi (default faster!)
    putstring_nl("Card init. failed!");  // Something went wrong, lets print out why
    sdErrorCheck();
    while(1);                           // then 'halt' - do nothing!
  }

  // enable optimize read - some cards may timeout. Disable if you're having problems
  card.partialBlockRead(true);
```

```
// Now we will look for a FAT partition!
  uint8_t part;
  for (part = 0; part < 5; part++) {     // we have up to 5 slots to look in
    if (vol.init(card, part))
      break;                             // we found one, lets bail
  }
  if (part == 5) {                       // if we ended up not finding one  :(
    putstring_nl("No valid FAT partition!");
    sdErrorCheck();      // Something went wrong, lets print out why
    while(1);                            // then 'halt' - do nothing!
  }

  // Lets tell the user about what we found
  putstring("Using partition ");
  //Serial.print(part, DEC);
  putstring(", type is FAT");
  //Serial.println(vol.fatType(),DEC);     // FAT16 or FAT32?

  // Try to open the root directory
  if (!root.openRoot(vol)) {
    putstring_nl("Can't open root dir!"); // Something went wrong,
    while(1);                            // then 'halt' - do nothing!
  }

  // Whew! We got past the tough parts.
  putstring_nl("Ready!");


//////////////////Setup for LCD
//setup lcd cols and rows
lcd.begin(16, 2);
//print out player names, play1 on row 0, play2 on row 1
lcd.print("Player1:    ");
lcd.setCursor(0,1);
lcd.print("Player2:    ");
displayScore();

// Intro song to game :)
playcomplete("THEME.WAV");
```

```
// Determmine game to play
avgInput();
int play = detMoveRPS(2);
if (play == PAPER) {
  game = IJ;
}
else {
  game = RPS;
}

// practice round
 playcomplete("tone2.WAV");

}

//////////////////////////////////////////////////
////////LOOP//////////////////////////////////
//////////////////////////////////////////////////
void loop() {
   // round initializing sounds
    playcomplete("tone1.WAV");
    delay(500);
    playcomplete("tone1.WAV");
    delay(500);
    playcomplete("toneup.WAV");
    delay(100);


   // MOVE -- take in analog input
    avgInput();

    // Bases on what game you're play, response will eary
    if (game == IJ) {
      //Determining what move was made
      int play1 = detMove(1);
      int play2 = detMove(2);
```

```
      // Play sound in Response
       playWhichSound(play1, play2);

      // Take score
      scoring(play1,play2);
    }


    else if (game == RPS) {
      int play1 = detMoveRPS(1);
      int play2 = detMoveRPS(2);

      scoringsounds(play1, play2);
    }
}


//////////////
///HELPERS///
//////////////


//////////////ANALOG INPUT AVERAGING/////////////////
void avgInput(){
    TS_avg_1 = 0;
    TS_avg_2 = 0;
    FS_ring_avg_1 = 0;
    FS_ring_avg_2 = 0;
    FS_ind_avg_1 = 0;
    FS_ind_avg_2 = 0;
    // average value over a sampling period
    for (int i=1; i <=100; i++){
        TS_avg_1 = (TS_avg_1 * ((float)i-1) + (float)analogRead(plyr1TSPin))/(float)i;
        TS_avg_2 = (TS_avg_2 * ((float)i-1) + (float)analogRead(plyr2TSPin))/(float)i;
        FS_ring_avg_1 = (FS_ring_avg_1 * ((float)i-1) +
(float)analogRead(plyr1RingPin))/(float)i;
        FS_ring_avg_2 = (FS_ring_avg_2 * ((float)i-1) +
(float)analogRead(plyr2RingPin))/(float)i;
```

```
        FS_ind_avg_1 = (FS_ind_avg_1 * ((float)i-1) +
(float)analogRead(plyr1IndPin))/(float)i;
        FS_ind_avg_2 = (FS_ind_avg_2 * ((float)i-1) +
(float)analogRead(plyr2IndPin))/(float)i;
        }
}


//////////////ACCELEROMETERS//////////////////////////
//Function Accel_Position will determine what position
//the accelerometer(tilt sensor) is in
//Argument analogin is analog read from accelerometer
//returns int 0 for up, 1 for down
int TS_Position(float analogin){
  if(analogin >= 150 && analogin <= 250){
    return UP;
  }
  else if (analogin >= 265 && analogin <=500){
    return DOWN;
  }
  else {
    return 3;
  }
}

//////////////FLEXING//////////////////////////
//Function flexing returns boolean
//if finger is bent or not
//Argument analoginFS is analog read from flex strip
bool flexing(float analoginFS){
  if (analoginFS <= 300){
    return OPEN;
  }
  else {
    return CLOSED;
  }
}

//Function detMove() will return an int
```

```
//which corresponds to gun, reload, shield move
//Args: int plyrTSPin corresponds to the player's
//tilt sensor pin on arduino, int plyrFSPin is the player's
//flex strip pin on arduino
int detMove(int plyr){
  //take the info from that players pin
  //if analog for down and bent, it is a gun
  if (plyr == 1) {
    if( TS_Position(TS_avg_1)==DOWN && flexing(FS_ring_avg_1)== CLOSED ){
    return GUN;
    }
    //if we have analog values for lifted up and bent, it is a reload
    else if( TS_Position(TS_avg_1)==UP && flexing(FS_ring_avg_1)== CLOSED ){
    return RELOAD;
    }
    //if analog for unbent, it is shield
    //NEED TO SPECIFY THIS IF BECAUSE IF PLAYER MADE WEIRD MOVEMENT
THEN IT COULD OUTPUT SHIELD WHEN WE DON'T WANT IT TO
    else if(flexing(FS_ring_avg_1)== OPEN ){
      return SHIELD;
    }
  }
  if (plyr == 2) {
    if( TS_Position(TS_avg_2)==DOWN && flexing(FS_ring_avg_2)== CLOSED ){
    return GUN;
    }
    //if we have analog values for lifted up and bent, it is a reload
    else if( TS_Position(TS_avg_2)==UP && flexing(FS_ring_avg_2)== CLOSED ){
    return RELOAD;
    }
    //if analog for unbent, it is shield
    //NEED TO SPECIFY THIS IF BECAUSE IF PLAYER MADE WEIRD MOVEMENT
THEN IT COULD OUTPUT SHIELD WHEN WE DON'T WANT IT TO
    else if(flexing(FS_ring_avg_2)== OPEN ){
      return SHIELD;
    }
  }

}
```

```
// determine move for Rock, Paper Scissors
int detMoveRPS(int plyr){
  //take the info from that players pin
  //if analog for down and bent, it is a gun
  if (plyr == 1) {
    if( flexing(FS_ring_avg_1)== CLOSED && flexing(FS_ind_avg_1)== CLOSED ){
    return ROCK;
    }
    //if we have analog values for lifted up and bent, it is a reload
    else if( flexing(FS_ring_avg_1)== CLOSED && flexing(FS_ind_avg_1)== OPEN ){
    return SCISSORS;
    }
    //if analog for unbent, it is shield
    //NEED TO SPECIFY THIS IF BECAUSE IF PLAYER MADE WEIRD MOVEMENT
THEN IT COULD OUTPUT SHIELD WHEN WE DON'T WANT IT TO
    else if(flexing(FS_ring_avg_1)== OPEN && flexing(FS_ind_avg_1)== OPEN ){
      return PAPER;
    }
  }
  else if (plyr == 2) {
    // both bent --> rock
    if( flexing(FS_ring_avg_2)== CLOSED && flexing(FS_ind_avg_2)== CLOSED ){
    return ROCK;
    }
    //ring finger bent, index unbent --> scissors
    else if( flexing(FS_ring_avg_2)== CLOSED && flexing(FS_ind_avg_2)== OPEN ){
    return SCISSORS;
    }
    //if all unbent --> paper
    //NEED TO SPECIFY THIS IF BECAUSE IF PLAYER MADE WEIRD MOVEMENT
THEN IT COULD OUTPUT SHIELD WHEN WE DON'T WANT IT TO
    else if(flexing(FS_ring_avg_2)== OPEN && flexing(FS_ind_avg_2)== OPEN ){
      return PAPER;
    }
  }

}
```

```
//////////////SOUND//////////////////////////
//Function playWhichSound() will play a sound
//on the speaker corresponding to which move was made
//Args: integers play1, play2 for the move player 1 & 2 made
void playWhichSound (int play1, int play2) {
       //CASE 1: if one hand is a gun
       if (play1==GUN || play2==GUN){
         playcomplete("GUN.WAV");
       }
           //CASE 2: if no guns, and one hand is reload
           else if(play1==RELOAD|| play2==RELOAD){
              playcomplete("RELOAD.WAV");
           }
             //CASE 3: if two shields
             else {
                playcomplete("SHIELD.WAV");
             }
}

// Plays a full file from beginning to end with no pause.
void playcomplete(char *name) {
  // call our helper to find and play this name
  playfile(name);
  while (wave.isplaying) {
  // do nothing while its playing
  }
  // now its done playing
}

void playfile(char *name) {
  // see if the wave object is currently doing something
  if (wave.isplaying) {// already playing something, so stop it!
    wave.stop(); // stop it
  }
  // look in the root directory and open the file
  if (!f.open(root, name)) {
    putstring("Couldn't open file "); Serial.print(name); return;
  }
  // OK read the file and turn it into a wave object
```

```
  if (!wave.create(f)) {
    putstring_nl("Not a valid WAV"); return;
  }

  // ok time to play! start playback
  wave.play();
}


//////////////////////////LCD//////////////////////////

//Function scoring will update and keep track
//of score of each player with vars: WIN1, WIN2
//based on move each player made
//Args: player1, player 2 are the moves player1 and 2 made
void scoring(int player1, int player2){
  // GUN, GUN case
  if (player1 + player2 == GUN + GUN) {
    // determine if either has store == 5 --> end game
    if (store1 >= 5 || store2 >= 5){
      if (store1 >= 5 && store2 < 5) {
        // Player 1 wins
        WIN1 += 1;      // increase WIN by 1
        endround();     // write code for what happens at the end of each round
      }
      else if (store1 < 5 && store2 >= 5) {
        // Player 2 wins
        WIN2 += 1;      // increase WIN by 1
        endround();
      }
      else {
        // Both blew each other up
        endround();
      }
    }
  }

  // GUN, RELOAD case
  else if (player1 + player2 == GUN + RELOAD){
```

```
  // player with gun wins
  if (player1 == GUN && store1 > 0) {
    // Player 1 wins
    WIN1 += 1;
    endround();
  }
  else if (player2 == GUN && store2 > 0) {
    // Player 2 wins
    WIN2 += 1;
    endround();
  }
}

// GUN, SHIELD case
else if (player1 + player2 == GUN + SHIELD){
  // determine which one had a gun
  if (player1 == GUN && store1 > 0){
    if (store1 >= 5) {
      // Player 1 wins
      WIN1 += 1;
      endround();
    }
    else {
      // game continues
      store1 = 0;
    }
  }
  else {
    if (store2 >= 5) {
      // Player 2 wins
      WIN2 += 1;
      endround();
    }
    else {
      // game continues
      store2 = 0;
    }
  }
}
```

```
  // RELOAD, RELOAD or RELOAD, SHIELD case
  else if (player1 + player2 == RELOAD + SHIELD || player1 + player2 == RELOAD +
RELOAD) {
    // if RELOAD, add to that players store
    if (player1 == RELOAD) {
      store1 += 1;
    }
    if (player2 == RELOAD) {
      store2 += 1;
    }
  }

  // SHIELD, SHIELD case
  else {
    // this is the shield all case so nothing...
  }
}

// scoring and sound fxn for rock, paper, scissors
void scoringsounds(int playera, int playerb) {

  // who won the round
  // same move -- play sound and continue
  if (playera + playerb == ROCK + ROCK || playera + playerb == PAPER + PAPER ||
playera + playerb == SCISSORS + SCISSORS){
    if (playera == ROCK){
      playcomplete("ROCK.WAV");
    }
    else if (playera == PAPER) {
      playcomplete("PAPER.WAV");
    }
    else {
      playcomplete("SCISSORS.WAV");
    }
  }

  // rock + paper
  else if (playera + playerb == ROCK + PAPER) {
```

```
  playcomplete("PAPER.WAV");
  if (playera == PAPER) {
    RPSWIN1 += 1;
  }
  else {
    RPSWIN2 += 1;
  }
}

// rock + scissors
else if (playera + playerb == ROCK + SCISSORS) {
  playcomplete("ROCK.WAV");
  if (playera == ROCK) {
    RPSWIN1 += 1;
  }
  else {
    RPSWIN2 += 1;
  }
}


// paper + scissors
else {
  playcomplete("SCISSORS.WAV");
  if (playera == SCISSORS) {
    RPSWIN1 += 1;
  }
  else {
    RPSWIN2 += 1;
  }
}

// Determine if the game was wone yet (best of 5) and if so, send to endround fxn
if (RPSWIN1 > 2) {
  WIN1 += 1;
  endround();
}
else if (RPSWIN2 > 2) {
  WIN2 += 1;
```

```
    endround();
  }
}



// fxn at the end of each round -- updates the LCD, clears variables, creates a delay for
players to regroup, and gives it the beat to restart
void endround() {
  // updates the LCD
  displayScore();

  // clears reload variables for next round
  store1 = 0;
  store2 = 0;

  RPSWIN1 = 0;
  RPSWIN2 = 0;

  // delay for regroup (via sound)
  playcomplete("THEMETTE.WAV");
  playcomplete("tone2.WAV");
  delay(400);

}



//Function displayScore will print score
//of each player onto LCD
void displayScore(){

  lcd.setCursor(10,0);
  lcd.print(WIN1,DEC);
  lcd.setCursor(10, 1);
  lcd.print(WIN2,DEC);
}
```