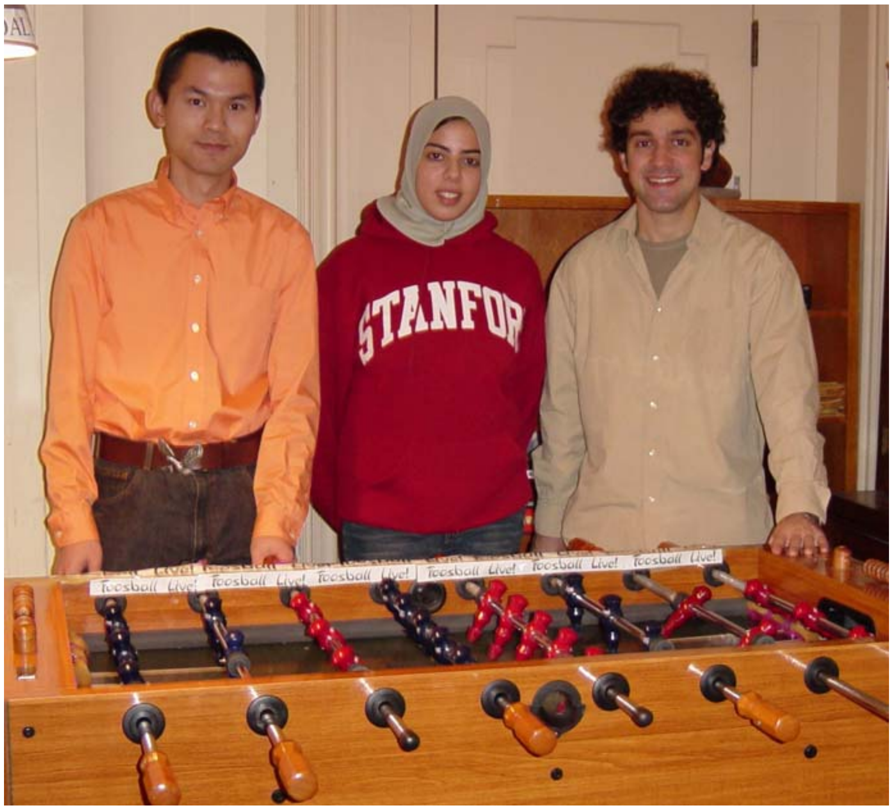
Foosball Live!

Design and Implementation of Human-Computer Interface



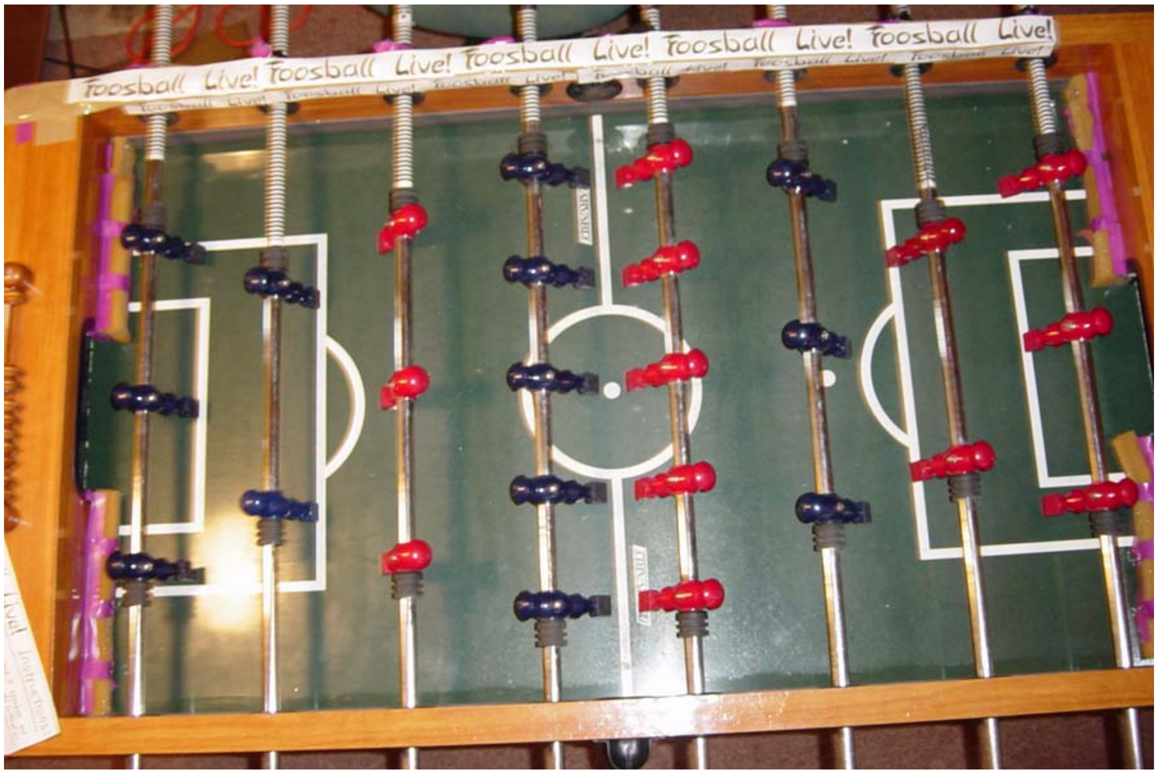
Rego Sen, Wai Kit Leung, Ariege Misherghi
Music 250a - Final Project
December 10, 2003

Soccer, more commonly known as football, is undoubtedly the most popular sport of the world, except in North America. It is therefore not surprising that the miniature form of the game, foosball, has also gained popularity around the world.

The idea behind our present project is that the foosball game is lacking the atmosphere of the real soccer game. We would like to incorporate crowd noise, real-time commentary, and other sound effects to liven the game play and to enhance the playing experience. The sound and music generated from game play should reflect the state of the game (i.e. by the level of action, score etc.) As we were requested to compose a piece of music using our interface, Appendix E has the score for a sample composition with its own notation legend. However, we intended this interface for chance music, and we feel that the result of natural game play is more interesting. *(See Appendix F for the eight-square interaction design sketch made in developing the concept of the interface.)*

Overview

The foosball table:


Foosball Live! Foosball Live! Foosball Live! Foosball Live! Foosball Live!

The foosball table in the Knoll Lobby at CCRMA has eight poles, four for each team. Each pole has a certain number of player pieces mounted on it, and the pole can slide across the table as well as rotated indefinitely, although by common consensus spinning the poles is disallowed in game play.

The first line of defense for each team has three player pieces, while the next line of defense has two. The five-man midfield is complemented by a three-man strike force.

The playing surface of the foosball table at CCRMA is level, unlike some that are slanted to the center.

On each side of the table there is a goal in the middle. A plastic tray collects the ball after a goal is scored and a plastic hose directs the ball to another tray on the side of the table for retrieval of the ball.

Rules
The team that first scores ten goals wins the match.

Implementation:
A. Sensory Information
- *1. Goal Detection (piezo discs)*
- *2. Player Action (optical sensors)*
- *3. Atmel interface and AVRmini coding*

B. Translation to Sound using Pure Data
- *1. Crowd Reaction*
- *2. Adaptive Music*
- *3. Real-time Commentary*
- *4. Victory Music*

A. Sensory Information

A1. Goal Detection
Scoring goals is the ultimate aim of playing. Spectators and commentators alike are rapturous each time a goal is scored. Therefore it is of paramount importance that goals in a foosball game be detected and that an appropriate event or chain of events be triggered as a result.

Alternatives considered:

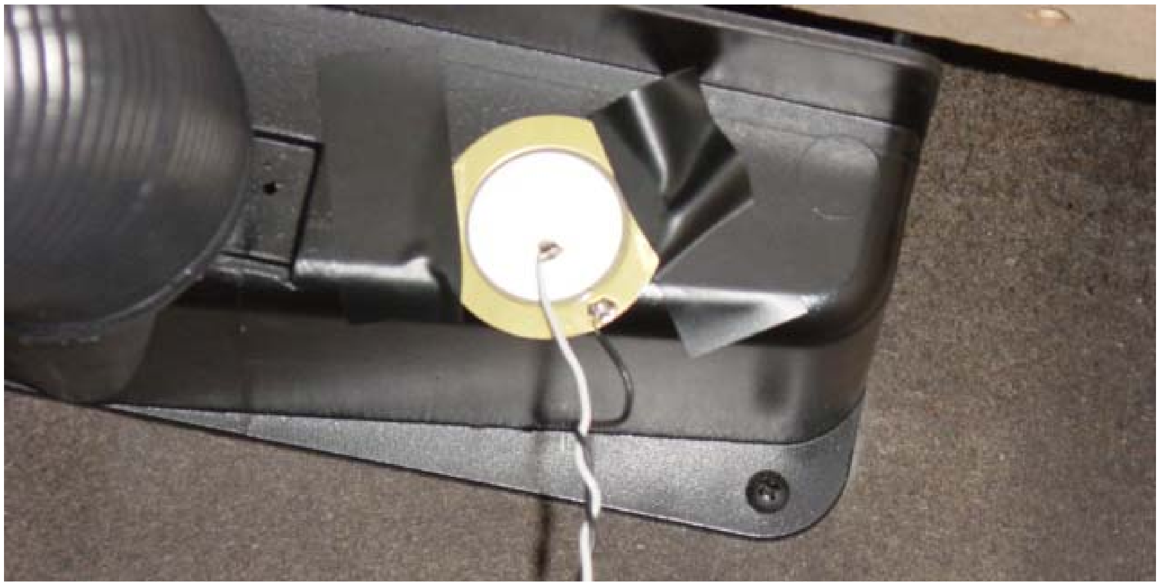
a) Laser beam across the face of the goal
A laser beam shooting across the face of the goal is effective detecting the ball going in (therefore blocking the laser beam momentarily). The drawback of this approach is the cost of the components, the interference to game play (the laser gun and the receiver will have to be placed somewhere along the byline), and the vulnerability of the components from getting hit by the ball.

b) Piezo or bend sensor installed on a screen on the goal
A lightweight screen with a piezo sensor or a bend sensor can be put on the goal, so that when the ball goes in the motion will be detected. The resistance of light weight paper was enough to keep a ball from entering the goal if it was moving a very slow rate. Such a huge bug would cause various unnecessary problems during game play.

c) Piezo or bend sensor in the gutter tubing underneath the table where the ball moved from goal side panel
Even having a single piezo strip barely crossing the path of the ball as it passed through the gutter would sometimes be enough to stop it completely and cause disruption in game play. In an attempt to keep the interface elegant, we threw out the option of using this alternative.

Solution used:

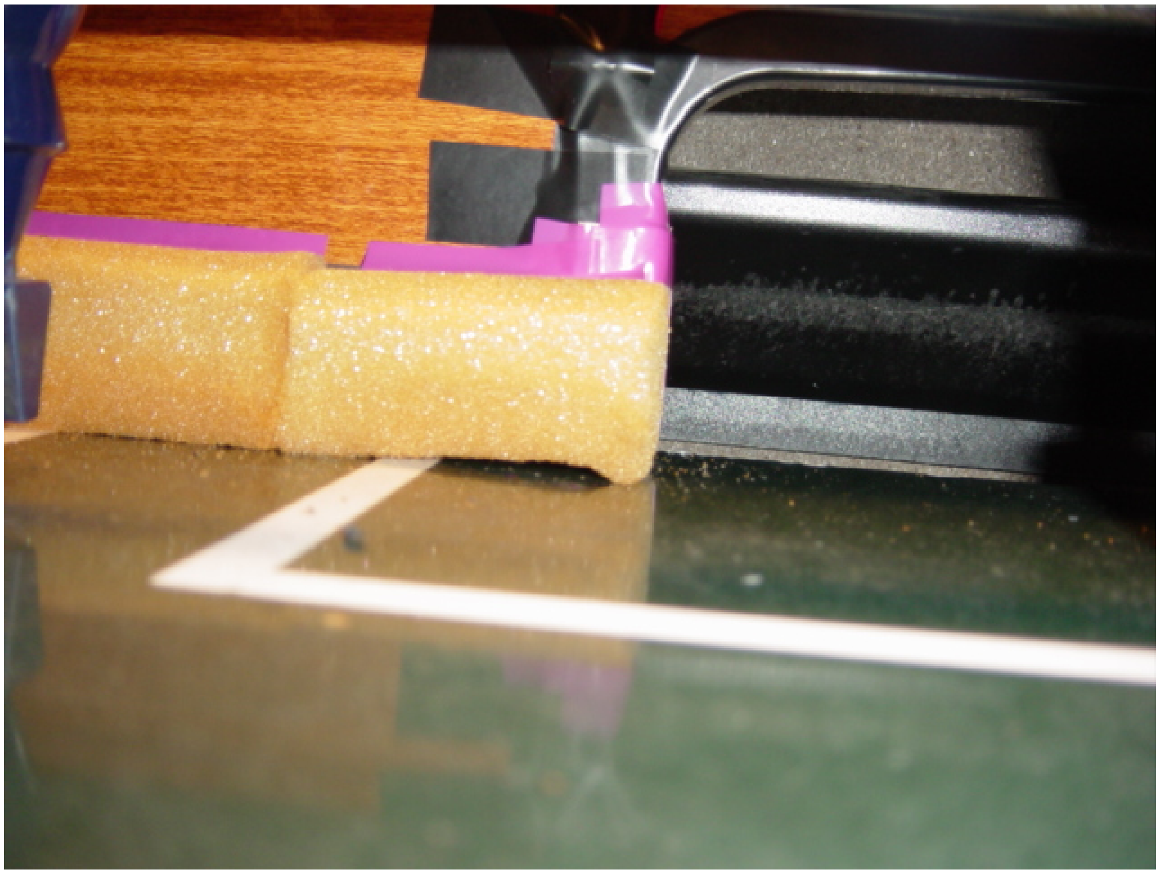
Piezo disc (microphone) installed on the tray behind the goal:



There is a distinct "ping" sound when a goal is scored either as a result of the ball hitting the back of the goal, or of the ball dropping to the bottom of the tray after rolling across the goal line. A piezo disc, which acts as a microphone, installed on the tray will pick up the ping sound and register a goal. We have programmed our Pd patch such that the goal detection is disabled for four seconds following the initial scoring of the goal, thus we prevent the possibility of multiple goals being triggered from the many "ping" sounds resulting from the foosball bouncing around in the goal.

Room for improvement:
The piezo disc picks up vibration (sound) and interprets it as a goal when the signal level is greater than a preset threshold value. It is therefore susceptible to noise. The main source of noise is noise from game play, especially when the ball hits the table hard, for example, a hard shot hitting the side of the goal. Although the piezo disc is installed on the goal tray, noise from something other than a goal does get picked up.

Solution:
Bumpers installed on the goal posts and on the sides of the goal.



They significantly dampen the sound of the ball hitting those areas and eliminated the majority of false triggers.

Future expansion:
More piezo discs can be installed at various locations on the table to pick up the local audio signal. The various signals can then be compared and the sound source located accordingly.

A2. Player Action
The motion of the player pieces, which is controlled by the poles, should generate music and/or sound appropriately. An increase in the level of activity of the player pieces should result in an increase in the intensity of the crowd noise.

Solution considered:
Velocity sensor
Velocity sensors (or alternatively displacement sensors) in the form of potentiometer are commercially available. Both the sliding type and the string-wound type introduce considerable resistance to the motion either in the form of sliding friction or string tension. Since there is a lot of movement in a foosball game, it is best to keep the resistance to motion to a minimum, if not zero. Also it is important to minimize the wear and tear on the parts.

Solution adapted:
Optical sensor
To reduce both the resistance to motion and the wear and tear to zero, a non-contact type way of sensing the velocity is desired. This was achieved with optical sensors. Black-and-white stripes were printed on labels that were then installed on the poles so that a photo sensor close to a pole would sense the alternating amount of light intensity (low intensity on black due to low reflection and high intensity on white due to higher reflection). The rate at which the light intensity changes gives the velocity of the pole.

Future expansion:
Labels were used for this project because they were temporary. They tend to wear out rather quickly by the intense action of the poles during game play. For permanent application the striped pattern should be applied with more durable labels or paint.

A3. Atmel Interface and AVRmini coding

Each optical sensor is wired to the Atmel breadboard using the schematic in Appendix A. Each output is then connected to an input port of the TI HC4052 multiplexer; we connected input 0 to one of the end poles, input 1 to the next pole, and so on. The GND, Vee, and É ports of the multiplexer were grounded, and the Vcc port was connected to +5V. We connected the A and B outs to the first and second inputs of Port A, respectively, and the S0 and S1 to the first and second outputs of Port B, respectively.

One end of each piezo sensor was grounded while the other end was connected to the third and fourth inputs of Port A. The third input was connected to the goal of the same team that controlled the optical sensor of the first input of the multiplexer.

The C code that was installed to the AVRmini is in Appendix C. The *include* files were provided by our instructors. Some aspects of the code are identified by commented lines. The rate of change of each sensor's output is calculated by taking the difference of the current and last readings. This value is then smoothed out using a first-order filter.

When a goal is scored, the piezo discs output an impulse from the bin vibration. Unfortunately, the impulse decays faster than one cycle of sensor readings, so we read the piezo data four times as often as the optical sensor data. Following the while statement are four subsections of similar- looking code. In each subsection the piezo outputs are read and filtered, and the S0/S1 bits of the multiplexer are changed to read the next two optical sensors. After these four subsections of code, all eight values of the optical sensors are filtered. The filtered sensor values are sent out of the serial port via OSC in the format (channel, value), where the sensors connected to multiplexer inputs 0-7 are assigned to channels 0-7, respectively. The piezo discs connected to inputs 3 and 4 of Port A are assigned to channels 8 and 9, respectively.

B. Translation to Sound using Pure Data

We wrote a Pd patch *(Appendices D1-3)* that inputs the OSC data and translates the information to sound. The optical sensor data was filtered out using the line~ patch over 1-second time frames. The resultant data was translated to data using the following ideas:

B1. Crowd Reaction
The soccer crowd, like the crowd for other games, is noisy, and noise level rises and falls according to the state of the game. Ideally, the crowd would react to goal scored, near misses, and change in possessions.

We used the Crowd Engine Pd patch written by Paul Leonard *(see Appendix D4)*. It uses a three-sample crossfading crowd model which uses the intensity of the crowd as the modulating input. The volume tables describe a crossfade such that it smoothly transitions between the low sample at low intensity and the high sample at high intensity. The pitch is also increased as the intensity increases to add more dynamics to the sound.

The movement of each pole adds to the crowd noise level and intensity. The crowd grows in pitch as the level of activities increases. In a real soccer game, the supporters of the two sides are separated. In order to make the playing experience more realistic, we made use of the possible stereo effects of having two loudspeakers, one on either side of the table. The movement of poles belonging to one team arouses reaction primarily from supporters of that team (i.e. more sound coming out from the loudspeaker representing supporters on that side).

Future expansion:
FSRs installed on the goal posts (and possibly other areas on the side of the goal) will detect the ball hitting those areas, thus triggering a "disappointing crowd" sound clip. Sensors installed on the goalkeeper can also sense the goalkeeping keeping out the goal and play back a "great save" sound clip, although in that case a well thought out algorithm must be implemented to distinguish goalkeeper saving a shot from goalkeeper kicking the ball.

B2. Adaptive Music
Apart from simulating sound effects of a real soccer game, we implement a soundtrack that is directly controlled by the poles. Each pole controls the playback rate of a sample so that as pole activity increases, the corresponding sample plays at a higher rate and amplitude. A given pole is assigned to a sample based on the score of its team. Each score corresponds to a bank of samples according to the following table:

Score	*Sample bank description*
0	bird sounds (taken from 220a homework 1 submissions)
1	acoustic guitar samples (performed by Danielle Smith)
2	piano samples (performed by Wilhelm Kempf)
3	Xylophone samples (from 192a text)
4	SATB a cappella samples (by Toby Twining)
5	Saxophone samples (from 192a text)

6	Accordion samples (from 192a text)
7	Percussion samples (assorted)
8	Swing samples (assorted)
9	Metallica samples

Each optical channel is routed to one of eight sub-patches corresponding to Appendix D2. While each subpatch is assigned to a different sample from each sample bank, the other parameters are the same.

B3. Real-time Commentary -- "Goal Sound"
The experience of watching a soccer match is always enhanced by the commentator's commentary. We are all too familiar with the passionate South American commentators' shout of "gooooooooooooooooal" ...
Sound clips from real game commentaries were found on a website of a fan of Leeds United (a team in the English Premier League). Excerpts were edited and converted from MP3 to wav using Audacity. The excerpts were padded with crowd noise to avoid an abrupt drop-out. For further excitement, another clip with a Spanish commentator shouting "goooooooal" apparently after a goal was scored, was also adapted.

A goal is triggered when the corresponding piezo output exceeds a given threshold.
In our main patch *(Appendix D1)*, the threshold was 110. We found this value to jointly minimize false triggers and untriggered goals.

When a goal is triggered, a value of 1 is sent to the goal subpatch *(see Appendix D3)*, which then plays one of four sound clips to enhance the excitement. A different clip (in round-robin fashion) is played each time to provide variety. As is the case for crowd reaction, we made use of stereo effect and the commentator's voice and the crowd's cheers come primarily from the side where the goal is scored.

Future expansion:
If speed of the ball (i.e. how hard the shot was taken), the flight of the ball and the location at which the ball hits the goal (right corner or left corner) can be accurately sensed, a dedicated clip (e.g. "wow, that's a screamer off the post") can be played back for more realistic commentary.

B4. Victory Music
When one side wins the game (i.e. score count reaches 10), it is appropriate to play an upbeat tune for victory. An excerpt of the 1812 Overture by Tchaikovsky is played when Team A wins and an excerpt of the William Tell Overture by Rossini is played when Team B wins.

Problems encountered:
1. Multiplexer
There were difficulties in getting the HC4052 multiplexer to work correctly.

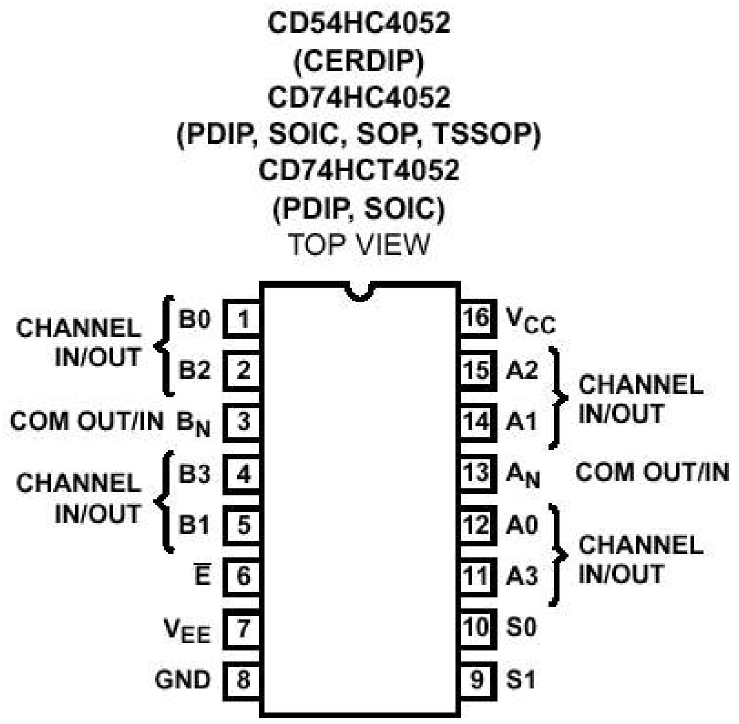
The problem turned out to be that the É port of the multiplexer had to be grounded but we had left it unconnected, which resulted in erratic behavior.

2. Atmel
We initially wired all eight optical sensors to the same power strip of the breadboard. This drew enough current to cause the power LED to dim greatly. We first attributed this to the erratic behavior of the multiplexer. Now we realize it may not have caused any implementation problems.

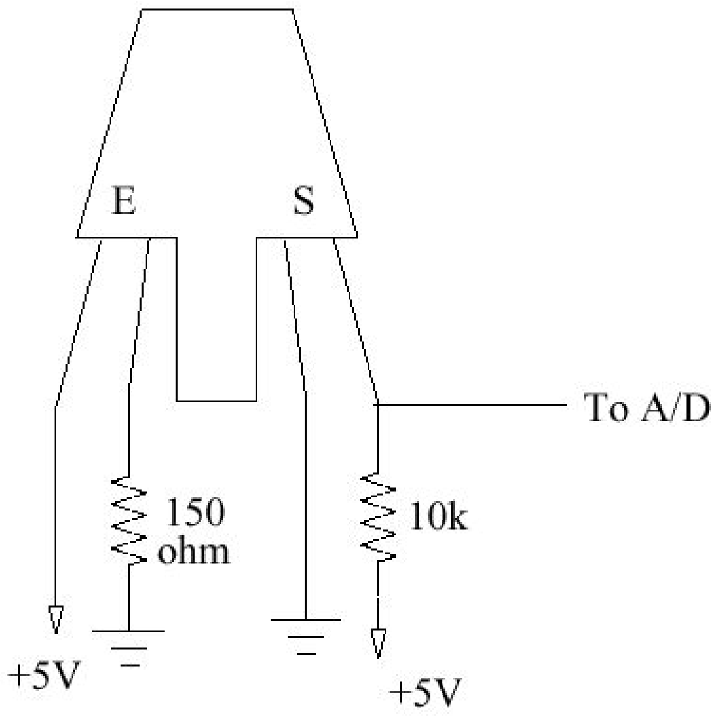
3. Sound Sampling
Difficulties were encountered in playing back wav files at the early stages of the project. Specifically, when a goal was scored there was a latency of about one second (during which a strange, electrical-like noise was generated) before the commentary clip could be played. The speed at which the clip was played was also inconsistent due to skipping.

It was thought that our Pd patches demanded excessive CPU usage and therefore adaptive music, score-reporting and crowd reaction were not implemented. Towards the latter stages of the project we switched to another computer (cmn51, an ATL) and the difficulties with wav files playback went away. The computer initially used (cmn6, a HP Vetra VL) was indeed too slow for this application. Crowd reaction was subsequently re-implemented but there was no time to implement the score-keeping (using recorded speech samples), which would have been straightforward since score was kept in the main Pd patch.

Appendix A: Wiring Schematic for HC4052 Multiplexer



Appendix B: Wiring Schematic for an Optical Sensor



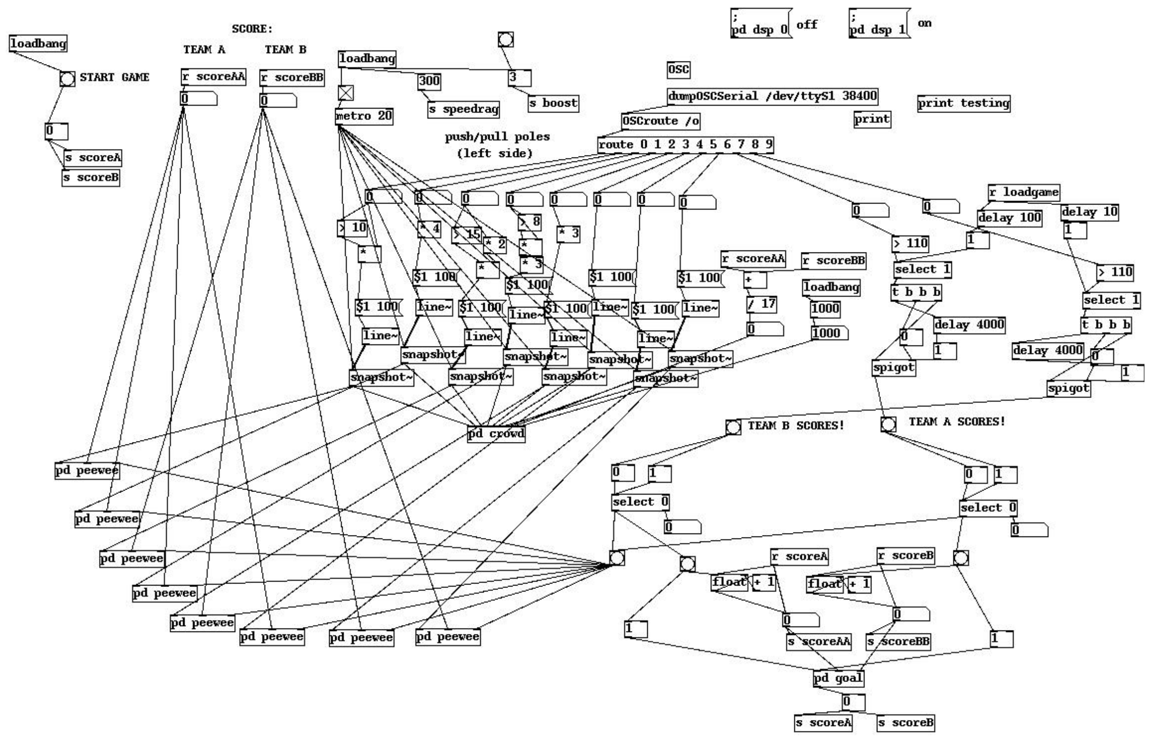
Appendix C: C Code for the AVRMini

```
//****************************************
//****************************************
//
//  FOOSBALL-LIVE.C
//  For avrlib and avrmini development board.
//
//
//  File:     foosball-live.c
//  Author:   Regaip Sen, Wai Kit Leung
//            some code by Michael Gurevich
//  Date:     Nov. 25, 2003
//
//
//  Updated for avrgcc3.x and atMega16
//  May 28, 2003
//
//
//****************************************
// This program sends osc messages out
// of the serial port.
//****************************************

#include <avr/io.h>
#include <avr/pgmspace.h>
#include <avr/interrupt.h>
#include <avr/signal.h>

#include "global.h"
#include "uart.h"
#include "ccrma/osc.h"
#include "ccrma/midi.h"
#include "timer.h"
#include "rprintf.h"
#include "a2d.h"

#define USEOSC // will use MIDI otherwise

int main(void)
{
  u16 a2dvalue[16];    // variable to store a/d value

  u08 i;

  s32 position[16];
  s32 lastpos[16];
  s32 velocity[16];
  s32 lastvel[16];
     u08 sdown;
     u08 tdown;

#ifdef USEOSC
     uartInit();
     oscInit();
     uartSetBaudRate(38400);
#else
     midiInit();
#endif
     timerInit();

     //     i = 0;
```

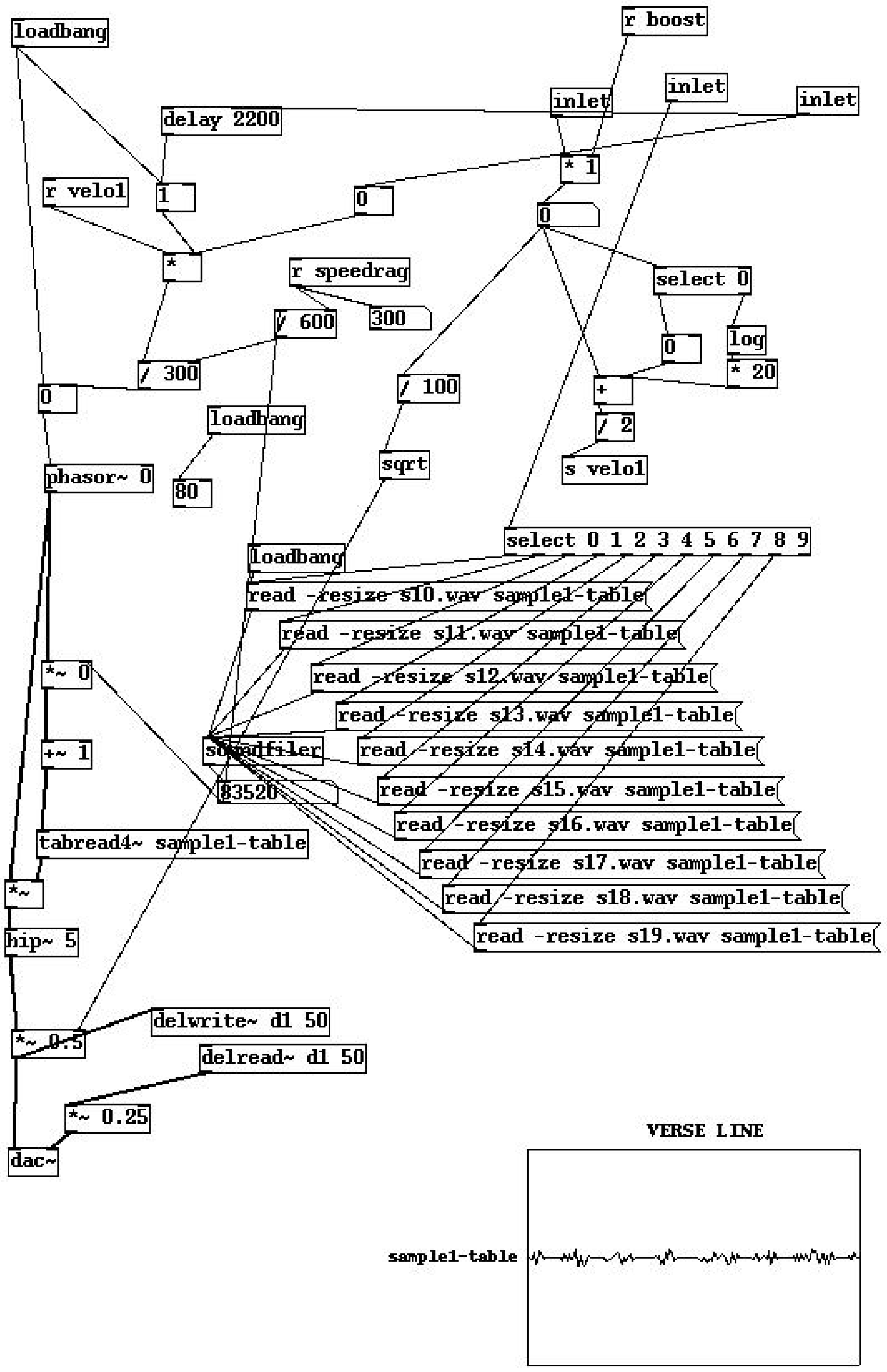
```
    for (i=0;i<16;i++) {
      position[i]=0;
    lastpos[i]=0;
    velocity[i]=0;
    }

  // ATOD STUFF
  a2dInit();
  a2dSetReference(0x01);

  sei();                   // enable interrupts
  outb(DDRA, 0xF0);        // set low 4 pins of Port A to input (for ADC)
                           // and high 4 pins to output (for LCD)
  outb(PORTA, 0x0F);       // set pull-ups to on
  outb(DDRB, 0xFF);

  while(1)
  {
    // CHANGE INPUT TO MULTIPLEXER, CHECK TWO SENSORS EACH TIME
    // SEND MIC DATA EACH TIME TO ENSURE DETECTION OF SHORT-DURATION
    // SIGNAL EMITTED BY BALL STRIKING GOAL BIN

      timerPause(5);                    // pause
      outb(PORTB, 0);
      a2dvalue[0] = a2dConvert10bit(0);
      timerPause(5);
      a2dvalue[4] = a2dConvert10bit(1);

      a2dvalue[8] = a2dConvert10bit(2);
      a2dvalue[9] = a2dConvert10bit(3);
      for (i=8;i<10;i++) {
      lastpos[i] = position[i];
      position[i] = a2dvalue[i];
      lastvel[i] = velocity[i];
      velocity[i] = velocity[i] + .5 * (((abs(position[i] - lastpos[i])) * 1)
- velocity[i]);
       if (lastvel[i] != velocity[i]) {
      oscSendMessageIntInt(PSTR("/o"), i, velocity[i]);
      }
      }
      timerPause(5);
      outb(PORTB, 1);
      a2dvalue[1] = a2dConvert10bit(0);
      timerPause(5);
      a2dvalue[5] = a2dConvert10bit(1);

      a2dvalue[8] = a2dConvert10bit(2);
      a2dvalue[9] = a2dConvert10bit(3);
      for (i=8;i<10;i++) {
      lastpos[i] = position[i];
      position[i] = a2dvalue[i];
      lastvel[i] = velocity[i];
      velocity[i] = velocity[i] + .5 * (((abs(position[i] - lastpos[i])) * 1)
- velocity[i]);
       if (lastvel[i] != velocity[i]) {
      oscSendMessageIntInt(PSTR("/o"), i, velocity[i]);
      }
      }

      timerPause(5);
      outb(PORTB, 2);
      a2dvalue[2] = a2dConvert10bit(0);
      timerPause(5);
```

```
        a2dvalue[6] = a2dConvert10bit(1);

        a2dvalue[8] = a2dConvert10bit(2);
        a2dvalue[9] = a2dConvert10bit(3);
        for (i=8;i<10;i++) {
        lastpos[i] = position[i];
        position[i] = a2dvalue[i];
        lastvel[i] = velocity[i];
        velocity[i] = velocity[i] + .5 * (((abs(position[i] - lastpos[i])) * 1)
- velocity[i]);
         if (lastvel[i] != velocity[i]) {
        oscSendMessageIntInt(PSTR("/o"), i, velocity[i]);
        }
        }

        timerPause(5);
        outb(PORTB, 3);
        a2dvalue[3] = a2dConvert10bit(0);
        timerPause(5);
        a2dvalue[7] = a2dConvert10bit(1);

        a2dvalue[8] = a2dConvert10bit(2);
        a2dvalue[9] = a2dConvert10bit(3);
        for (i=8;i<10;i++) {
        lastpos[i] = position[i];
        position[i] = a2dvalue[i];
        lastvel[i] = velocity[i];
        velocity[i] = velocity[i] + .5 * (((abs(position[i] - lastpos[i])) * 1)
- velocity[i]);
         if (lastvel[i] != velocity[i]) {
        oscSendMessageIntInt(PSTR("/o"), i, velocity[i]);
        }
        }
        // CALCULATE RATE OF CHANGE IN OPTICAL SENSORS
        for (i=0;i<8;i++) {
        lastpos[i] = position[i];
        position[i] = a2dvalue[i];
        lastvel[i] = velocity[i];
        velocity[i] = velocity[i] + .5 * (((abs(position[i] - lastpos[i])) * 1)
- velocity[i]);
        if (velocity[i] < 3) { velocity[i] = 0; }
        }

#ifdef USEOSC

        // SEND OPTICAL SENSOR DATA VIA OSC

        for (i=0;i<8;i++) {
         if (lastvel[i] != velocity[i]) {
        oscSendMessageIntInt(PSTR("/o"), i, velocity[i]);
         }
        }

#else
        midiNoteOnOut(a2dvalue1/8, a2dvalue2/10, 1);

#endif

  }
    a2dOff();
    return 0;
}
```

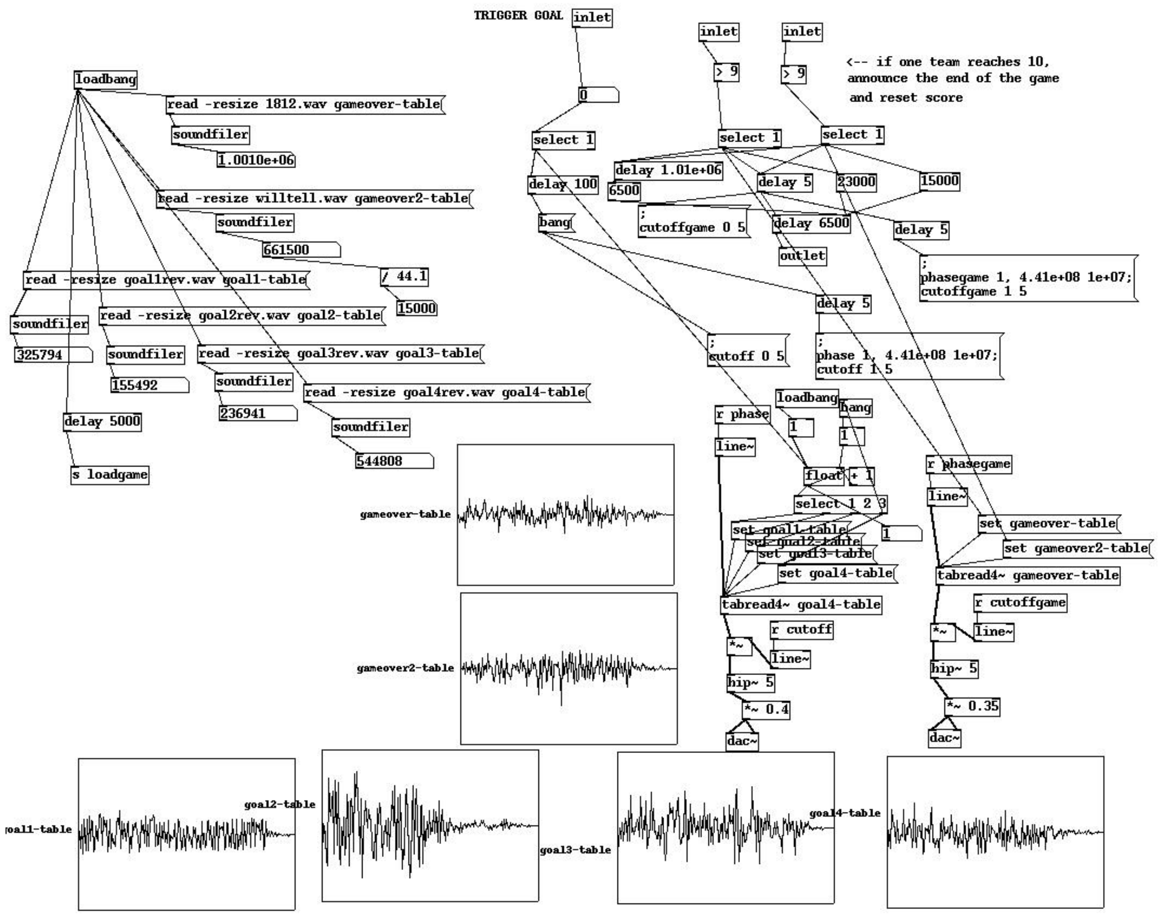
Appendix D1: Screen Shot of Main PD Patch



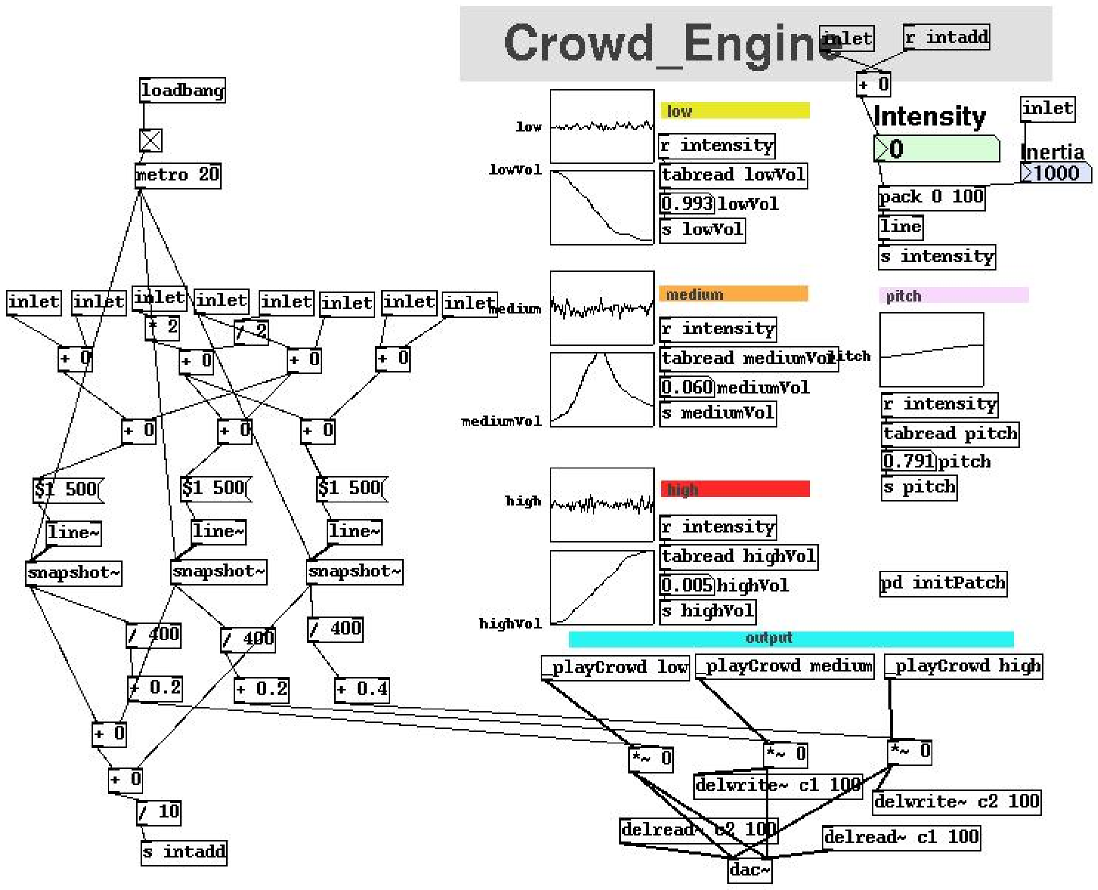
Appendix D2: Screen Shot of an Individual Pole's PD Patch


loadbang
r boost
inlet
inlet
inlet
delay 2200
* 1
r velo1
1
0
0
*
r speedrag
select 0
/ 600
300
0
log
* 20
/ 300
/ 100
+
0
loadbang
/ 2
phasor~ 0
80
sqrt
s velo1
select 0 1 2 3 4 5 6 7 8 9
loadbang
read -resize s10.wav sample1-table
read -resize s11.wav sample1-table
*~ 0
read -resize s12.wav sample1-table
read -resize s13.wav sample1-table
+~ 1
soundfiler
read -resize s14.wav sample1-table
83520
read -resize s15.wav sample1-table
read -resize s16.wav sample1-table
tabread4~ sample1-table
read -resize s17.wav sample1-table
*~
read -resize s18.wav sample1-table
read -resize s19.wav sample1-table
hip~ 5
delwrite~ d1 50
*~ 0.5
delread~ d1 50
*~ 0.25
VERSE LINE
dac~
sample1-table

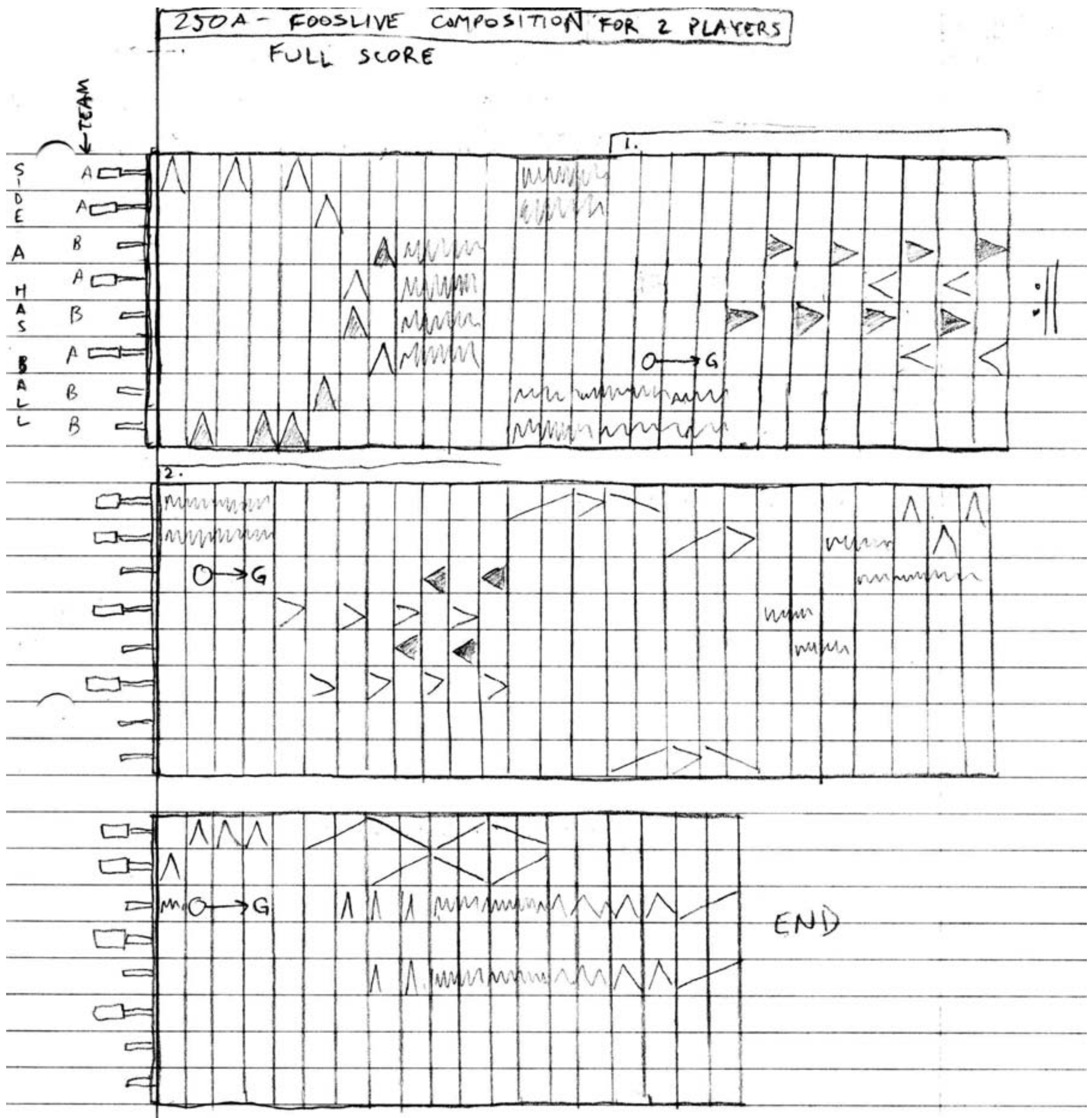
Appendix D3: Screen Shot of the Goal PD Patch



Appendix D4: Screen Shot of the Crowd Engine PD Patch (Modifications on left side of the screen)



Appendix E: Composition for Foosball Table (Full Score and Parts)



NOTATION:

$\boxed{\Lambda}$ = twist ccw, then cw $\boxed{V}$ = twist cw, then ccw

$\boxed{>}$ = push, then pull $\boxed{<}$ = pull, then push

$\boxed{ww}$ = trill-like motion: twist and push/pull simultaneously

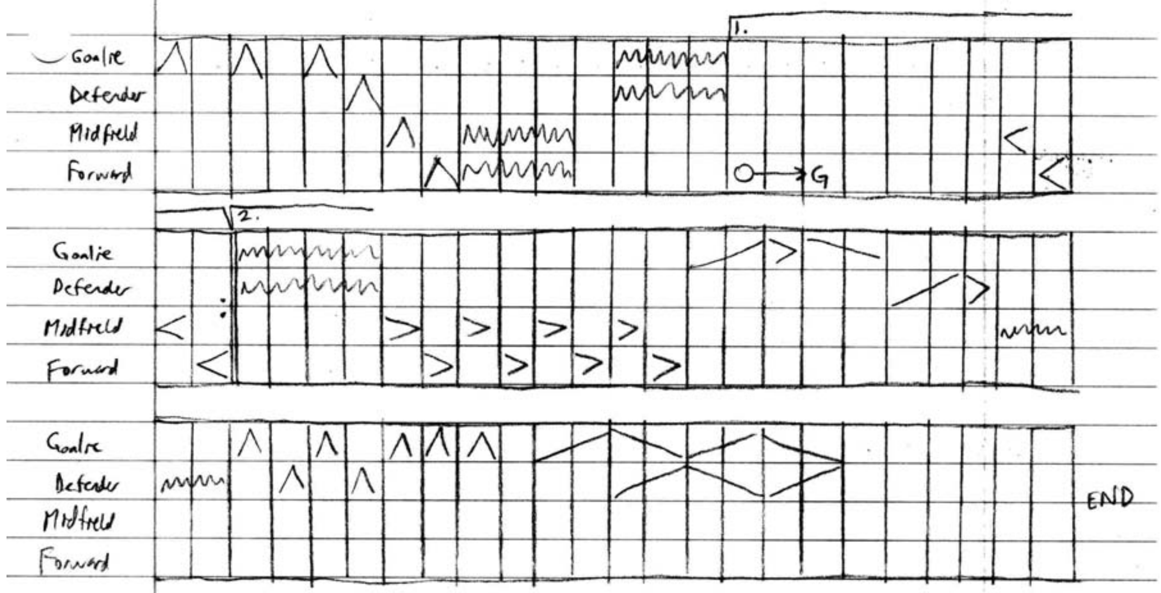
$\boxed{O \rightarrow G}$ = 1. place ball in front of center player piece

2. score goal against other team

3. set ball on table top

250 A – FOOSLIVE COMPOSITION FOR 2 PLAYERS

TEAM A SCORE



NOTATION:

$\boxed{\wedge}$ = twist CCW then CW

$\boxed{\vee}$ = twist CW then CCW

$\boxed{>}$ = push, then pull

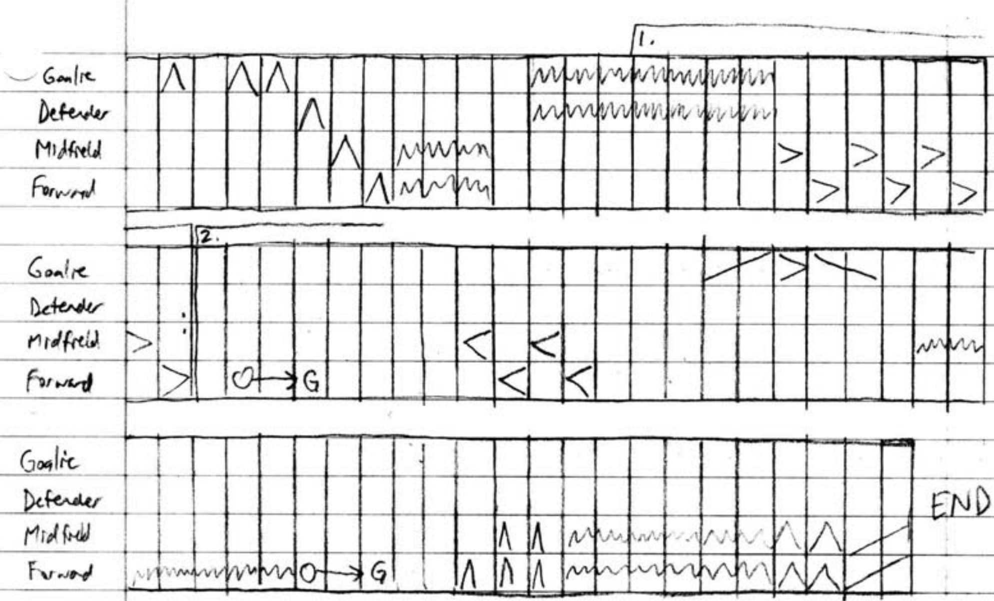
$\boxed{<}$ = pull, then push

$\boxed{\sim}$ = trill-like motion: twist and push-pull simultaneously

$\boxed{O \rightarrow G}$ = 1. place ball in front of center player piece
2. score goal against other team
3. set ball on table top

250A - FOOSLIVE COMPOSITION FOR 2 PLAYERS

TEAM B SCORE



NOTATION:

$\boxed{\Lambda}$ = twist ccw then cw

$\boxed{V}$ = twist cw then ccw

$\boxed{>}$ = push, then pull

$\boxed{<}$ = pull, then push

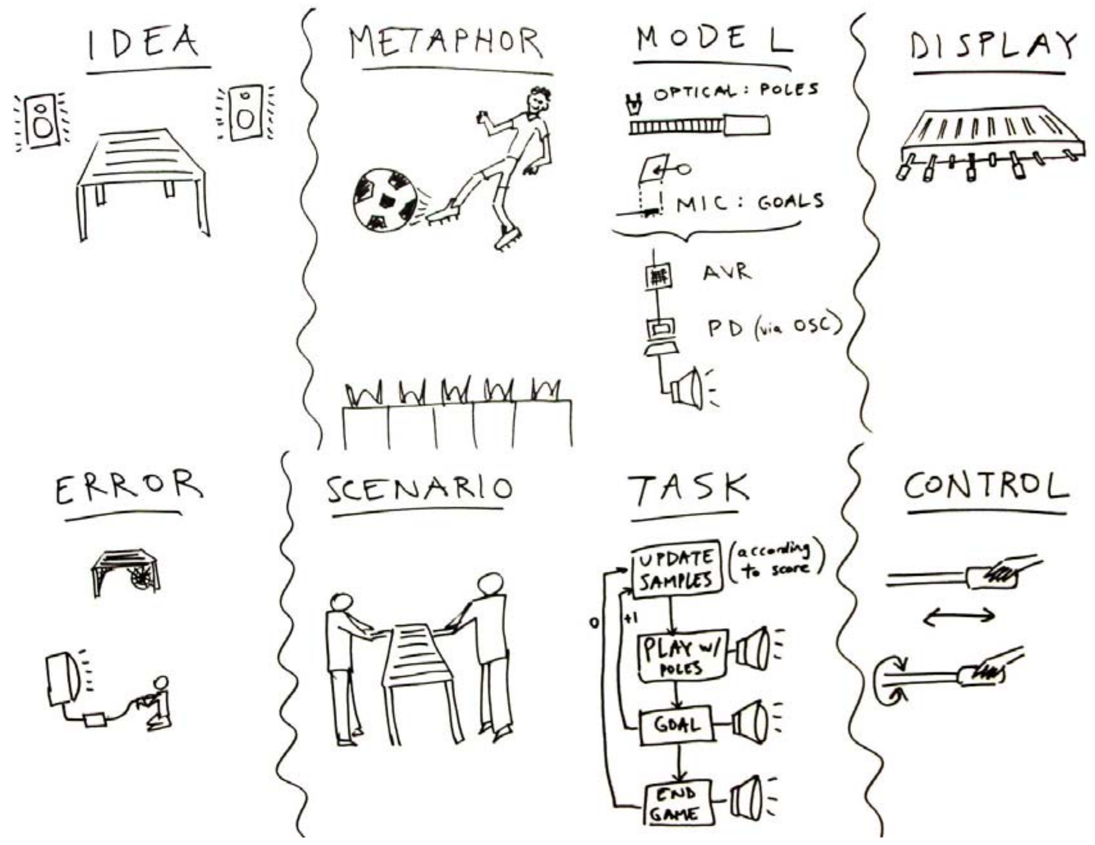
$\boxed{\sim}$ = trill-like motion: twist and push/pull simultaneously

$\boxed{O \rightarrow G}$ = 1. place ball in front of center player piece

2. score goal against other team

3. set ball on table top

Appendix F: Eight-Square Interaction Design Sketch



Description:

IDEA: Foosball game with audio

METAPHOR: Actual soccer game

ERROR: Video sports games are more interesting than traditional foosball

SCENARIO: People playing foosball as they would traditionally

MODEL: Sensors connected to poles and goal post sending information through AVRmini to PD implementation and output as audio.

DISPLAY: Same as foosball table

TASK: Sounds made when poles moved, goal scored and game ended. Score changes with goals.

CONTROL: push, pull, and twist poles.