

Christina Wolfskill

Dan Maples

Matthew Hilgers

CSCI 4830 – Things That Think

5/1/08

Smart Blocks Project Write-up

Introduction

The Things That Think final project was to create an interactive construction kit. The goal was for the project to have many permutations, not just one proper way of being put together. It also had to incorporate some sort of computation. The team decided to focus on creating a project that would be educational for children; after brainstorming, three possible projects concepts were formed. The first project involved building bridges using interconnecting pieces; one of the pieces would be fitted with a strain gauge to measure force. Once a bridge had been fully constructed, the child could then press down on it until it failed; a screen would then output the amount of force the bridge could with stand prior to collapsing. The second project idea used gears, which a child could put together with the goal of producing a specific gear ratio. The gears would then be used to move a car on a track or some other visual aid that would show the effect of adding more gears and different size gears. Also, shaft encoders would be used to output the actual gear ratio. The final project idea was a base station into which number blocks could be placed to form equations. The base station would read the equation and give feedback to the user as to whether it is correct or incorrect. This final concept was chosen because it seemed to be very feasible to construct, and the same concept could be adapted for many other uses including spelling, word associations, or colors. Thus, it would make a great prototype that would showcase a highly adaptable concept.

Original Concept

The original concept involved number blocks and operator blocks could be arranged into simple arithmetic equations. For the prototype, it was decided to simplify the design by only allowing for two operand number blocks, one operator block, an equals block and an answer block, thus limiting the equations to those containing only one digit numbers. Two possibilities were discussed for differentiating between the blocks: either resistors of different sizes or, preferably, RFID tags would be placed under each block. The team quickly decided on RFID tags because there would be more possible denominations (to allow for more distinct blocks). The bottom of each block would also have a permanent magnet. The base station would have an RFID reader underneath each slot to determine which number or operator was being placed into that slot. If the equation being put together was incorrect, an electromagnet beneath the last block would turn on, repelling the magnet in the bottom of that block to provide feedback to the user that this is not a correct equation. The computation was to be done using an Arduino microcontroller (shown in Figure 1), and the software would be implemented using the native Arduino programming language, which is a variation of C.

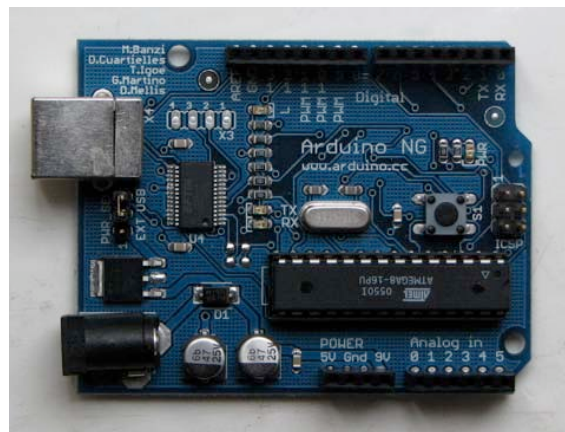


Figure 1: Arduino Microprocessor

Problems Encountered and Design Changes

We did run into a few problems while programming the Arduino and RFID reader. It was first discovered that only one RFID reader could be used with the Arduino. This is because the RFID reader communicates over serial, and the Arduino only has one UART. As a result, the reader was placed onto a slider with a handle, so that it could read the tags on all five blocks. Figure 2 shows the RFID reader on its sliding mechanism.

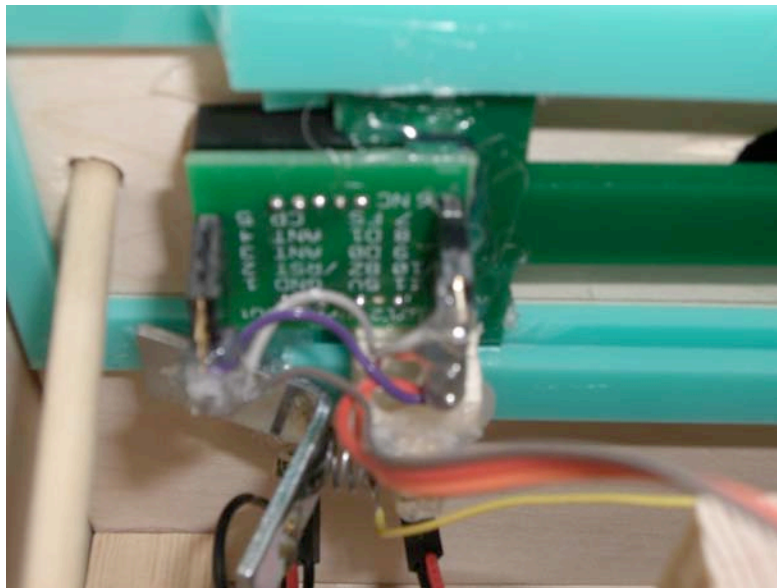


Figure 2: RFID Reader with Acrylic Slider

When we first got started using the RFID reader and Arduino we ran into a strange problem. For some reason the RFID reader was not sending data properly to the Arduino or anything else we tried to connect it to. After an afternoon of frustration with Nwanua, the problem mysteriously went away and never resurfaced. Also, the serial line from the RFID reader had to be physically disconnected from the Arduino when uploading programs; otherwise it would give upload errors. Another problem we encountered manifested itself in many bizarre ways, but we ultimately figured it out. When declaring constants for the known tag numbers, we had initially not included null values at the end of each array. This caused all sorts of seemingly random and disconnected errors, but after we included the null values, all the problems went away.

Another major design change was in the way that the feedback is given to the user. We originally wanted to use a magnet to pop the answer block out of the base station if it was not the correct block. However, after doing a little bit of research and observing problems other groups had while using magnets with the Arduino, we decided to find another mechanism to perform this part of the project. After some brainstorming, we decided to hook up a 5 Volt motor to a push rod. If the answer was wrong, the motor would turn on and move the pushrod through a hole in the base piece and hit the answer block. As the motor continues to turn, the rod rapidly goes up and down, causing the block to vibrate.

Software

IDE Overview

The Arduino software and IDE are well designed and make it very easy to program, especially for somebody that knows how to program in C/C++. The programming language that the Arduino microcontroller uses is a subset of C/C++; therefore, if you already know even a little bit of C/C++, programming for the Arduino is a breeze. Figure 3 shows an example of the Arduino programming environment.



Figure 3: Arduino Programming Environment

When compiling, the Arduino IDE modifies the code slightly, adding things like function prototypes and library declarations, and then passes it to a normal C/C++ compiler to compile. This whole process is automated, making it very easy to develop, test, and deploy code. The IDE became a little bit slow at times, but this was more of an annoyance than a serious problem. The only major problem we ran into with the IDE was that it froze the machine we were working on a few times.

Final Program Overview

As in all Arduino programs, the program loops in a function named `loop()`. In `loop()`, our code spin locks on `Serial.available()` checking if there is serial data waiting. When a tag is read, the RFID reader sends the data over the serial line to the Arduino. The spin lock opens and the serial data is read into a variable called `temp`. `temp` is then passed to `getTag()` to decode the data and return a tag number without the start, stop, or checksum bits. This is then passed to the `isEqual()` function to compare it to a list of known tags to identify which block was just read. Then, an integer value is placed in a queue by the function `addUnit()`, the actual integer value if it is a number, or a special constant number for an operator or equals sign. After each read, the program calls `calc()` to check the queue to see if the last five blocks form a valid equation. If there is a valid equation, the program lights up the green LED,

indicating that the equation is valid and correct. If the equation isn't valid, or the answer is incorrect, the red LED will illuminate and the motor will turn on to vibrate the answer block, giving the user feedback in two different forms. The code written for the project can be found in the Appendix.

Hardware

During construction, we decided to build most of the apparatus out of wood. We made a solid top piece with a rectangular recessed area cut out of it. This is where the number blocks and operator blocks sit to form a complete equation. A laser cutter was used to etch patterns into the sides of the base station for aesthetics. The slider for the RFID reader is made out of acrylic that was cut using the laser cutter. Other materials include a wooden dowel for vibrating the block, a wire to connect the dowel to the motor, screws to mount the Arduino, and hot glue to put everything together. The number blocks were drawn up using SolidWorks and then printed using the rapid prototype. Currently, eight number blocks have been created, the numbers 1 through 5, an additional 1, a plus, and an equal sign.

Finished Product

Figure 4 shows a photograph of the completed project from the front, and Figure 5 shows the backside of the project.



Figure 4: Completed Smart Blocks Project

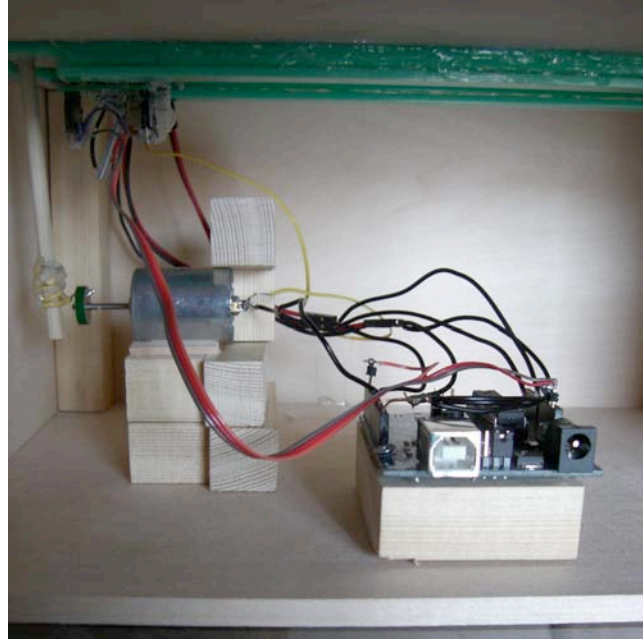


Figure 5: Working Mechanisms of the Project

Figure 6 shows a basic schematic diagram with the major components of the project.

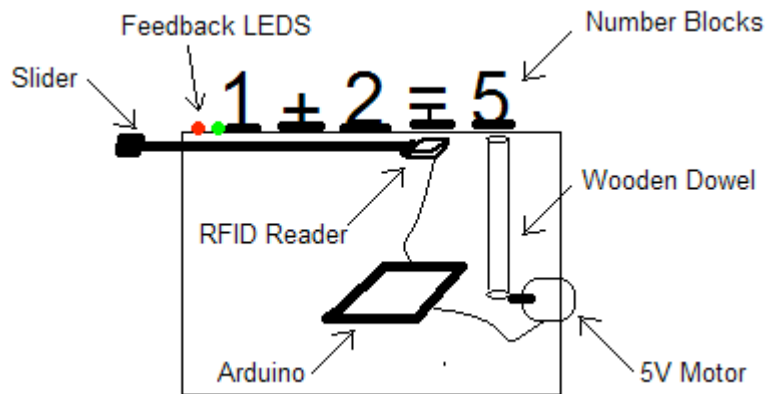


Figure 6: Schematic Diagram of the Project

Potential Applications

The Smart Blocks project was designed as a prototype that could be taken to toy companies to show the basic functionality of a potential product. The project was designed to be highly modifiable such that the same basic premise could be adapted to other similar objectives. The concept was shown for equations, but could be adapted to be used for letters, spelling words, or for word associations, or even as an interactive version of the game SET. The goal would be for this to be implemented as an educational toy for children who are kinesthetic learners, or as an adaptable learning tool for classrooms. Different sets of pieces could be sold to go with the same base station that would allow it to be used to teach children many topics, from math to spelling to colors, among others. It also does

not necessarily need to be constrained to a linear application. By placing RFID readers at different locations on the base station, the toy could relate objects in 2D space as well for applications such as geography, or to show which one of a set of objects does not belong. These stated applications are only a small subset of the many potential uses for this type of toy.

Future Work

If given three more weeks to further develop the project, more number blocks would have been created. The software can currently support digits up through 9, as well as subtraction, multiplication, and division. Thus, all that would be necessary in order to implement these functions would be to create the blocks in SolidWorks and print them. Also, with three more weeks, the design could be refined such that it would give less incorrect results and would slide more smoothly.

If given 3 more months to work on this project, the group would try to implement some of the other possible functions such as using letters to spell words, or playing the game SET. Also, ideally a different microcontroller would be found that could support multiple RFID readers, allowing for functionality more like that of the original design.

Conclusion

Smart Blocks are still in the initial prototype phase, but this project displays their functionality for providing feedback as to whether a mathematical equation is right or wrong. This is merely a small subset of the potential applications of this type of educational toy. Using the same concept of blocks with embedded RFID tags and a base station to read the tags and evaluate them, children could be taught many different subjects. The toy would be particularly useful for children who are kinesthetic learners, a type of learning style that is not often targeted in classrooms.

Appendix – Arduino Code

```
/*
 * Smart Blocks Code
 * Dan Maples
 * Dan(dot)Maples(at)colorado(dot)edu
 * April, 2008
 * RFID code uses example code from:
 * Alexander Reeder, Nov 16, 2007
 * and
 * Brian Riley, January 2008
 */
#define RESETLEDPin 13
#define RESETPIN 2
#define plus 10
#define minus 11
#define multiply 12
#define divide 13
#define equalsC 14

int equation[6];
char zero[11];
char one[11];
char one2[11];
char two[11];
char three[11];
char four[11];
char five[11];
char six[11];
char seven[11];
char eight[11];
char nine[11];
char plusTag[11];
char minusTag[11];
char multiplyTag[11];
char divideTag[11];
char equalsTag[11];
boolean motor = false;
long k=1;
```

```

char *getTag()
{
    char val = 0;
    static char IDstring[13];
    int i;
    if (Serial.available() > 0 )
    {
        delay(20);
        val = Serial.read();
        if ( val == 02 )
        { // look for Start Of Text marker
            for ( i = 0; (val = Serial.read()) != 03 ; i++)
            {
                IDstring[i] = val;
            }
            IDstring[10] = 0x00; // tie off IDstring at the CR-LF
            resetID12(); // reset after a valid read
        }
    }
    return IDstring;
}

boolean isEqual(char in[10], char con[10])
{
    for(int i =0;i<11;i++)
    {
        if(in[i]!=con[i])
        {
            return false;
        }
    }
    return true;
}

boolean calc(int in[6])
{
    if(in[0]<0 || in[0]>9) //check first slot for number
    {
        Serial.println("not a valid equation, left side of the equation not a number");
        return false;
    }
    if(in[1]<plus || in[1]>equalsC)//check second slot for operator
    {
        Serial.println("not a valid equation, operator not recognized");
    }
}

```

```

    return false;
}
if(in[2]<0 || in[2]>9) //check second slot for number
{
    Serial.println("not a valid equation, right side of equation not a valid number");
    return false;
}
if(in[3]!=equalsC) //check for equals sign
{
    Serial.println("equals sign not found");
    return false;
}
if(in[4]<0 || in[4]>9) //check answer slot for number
{
    Serial.println("not a valid equation, answer slot not a number");
    return false;
}
if(in[1]==plus)
{
    if(in[4]!= in[0] + in[2])
    {
        Serial.println("valid equation, wrong answer");
        return false;
    }
    else
    {
        Serial.println("correct!");
        return true;
    }
}
else if(in[1]==minus)
{
    if(in[4]!= in[0] - in[2])
    {
        Serial.println("valid equation, wrong answer");
        return false;
    }
    else
    {
        Serial.println("correct!");
        return true;
    }
}
}

```

```

}
else if(in[1] == multiply)
{
    if(in[4] != in[0] * in[2])
    {
        Serial.println("valid equation, wrong answer");
        return false;
    }
    else
    {
        Serial.println("correct!");
        return true;
    }
}
else if(in[4] != in[0] / in[2])
{
    Serial.println("valid equation, wrong answer");
    return false;
}
else
{
    Serial.println("correct!");
    return true;
}
return false;
}

void associateTags()
{
one[0]=48;one[1]=49;one[2]=48;one[3]=54;one[4]=53;one[5]=69;one[6]=57;one[7]=65;one[8]=52;one[9]=48; one[10]=0x00;
    one2[0]= 49;one2[1]= 49;one2[2]= 48;one2[3]= 48;one2[4]= 51;one2[5]= 69;one2[6]= 52;one2[7]= 67;one2[8]= 51;one2[9]= 53;one2[10]=0x00;
    two[0]= 48;two[1]=49 ;two[2]=48 ;two[3]=54 ;two[4]=53 ;two[5]= 69;two[6]=56 ;two[7]= 69;two[8]= 69;two[9]= 53; two[10]=0x00;
    three[0]= 48;three[1]= 49;three[2]=48 ;three[3]=54 ;three[4]=53 ;three[5]=69 ;three[6]=57 ;three[7]=68 ;three[8]= 55;three[9]= 70;three[10]=0x00;
    four[0]= 48;four[1]= 49;four[2]= 48;four[3]= 54;four[4]= 53;four[5]= 69;four[6]= 66;four[7]= 55;four[8]= 65;four[9]= 70;four[10]=0x00;
    five[0]= 48;five[1]= 49;five[2]= 48;five[3]= 54;five[4]= 53;five[5]= 69;five[6]= 66;five[7]= 65;five[8]= 50;five[9]= 54;five[10]=0x00;
}

```

```

    equalsTag[0]= 49;equalsTag[1]=49 ;equalsTag[2]= 48;equalsTag[3]= 48;equalsTag[4]= 51;equalsTag[5]=
69;equalsTag[6]= 52;equalsTag[7]= 67;equalsTag[8]= 56;equalsTag[9]= 66;equalsTag[10]=0x00;
    minusTag[0]= 48;minusTag[1]= 70;minusTag[2]= 48;minusTag[3]= 48;minusTag[4]= 48;minusTag[5]=
70;minusTag[6]= 49;minusTag[7]= 54;minusTag[8]= 49;minusTag[9]= 68;minusTag[10]=0x00;
    plusTag[0]= 49;plusTag[1]= 49;plusTag[2]= 48;plusTag[3]= 48;plusTag[4]= 51;plusTag[5]=
69;plusTag[6]= 56;plusTag[7]= 55;plusTag[8]= 56;plusTag[9]= 66;plusTag[10]=0x00;
}
int* addUnit(int in)
{
    equation[0]=equation[1];
    equation[1]=equation[2];
    equation[2]=equation[3];
    equation[3]=equation[4];
    equation[4]=in;
    equation[5]=0x00;
return equation;
}
void resetID12()
{
    digitalWrite(RESETLEDPin, HIGH); // show reset by lighting LED
    digitalWrite(RESETPIN, LOW); // pull reset down
    delay(100);
    digitalWrite(RESETLEDPin, LOW); // Shut off LED
    digitalWrite(RESETPIN, HIGH); // pull up Reset line
}
void setup() {
    associateTags();
    Serial.begin(9600); // connect to the serial port
    pinMode(RESETPIN, OUTPUT); // sets the digital pin as output
    pinMode(RESETLEDPin, OUTPUT); // sets the digital pin as output
    pinMode(12,OUTPUT);
    pinMode(8,OUTPUT);
    pinMode(5,OUTPUT);
    pinMode(10,OUTPUT);
    digitalWrite(10,HIGH);
    equation[5]=0x00;
}
void loop () {
    digitalWrite(RESETLEDPin, LOW); // Shut off Resest LED
    digitalWrite(RESETPIN, HIGH); // pull up Reset line
    if(Serial.available(>0)
    {

```

```
char* temp;
temp=getTag();
Serial.print("temp:");
Serial.println(temp);
if(isEqual(temp,one))
{
  addUnit(1);
}
if(isEqual(temp,one2))
{
  addUnit(1);
}
else if(isEqual(temp,two))
{
  addUnit(2);
}
else if(isEqual(temp,three))
{
  addUnit(3);
}
else if(isEqual(temp,four))
{
  addUnit(4);
}
else if(isEqual(temp,five))
{
  addUnit(5);
}
else if(isEqual(temp,plusTag))
{
  addUnit(plus);
}
else if(isEqual(temp,minusTag))
{
  addUnit(minus);
}
else if(isEqual(temp, equalsTag))
{
  addUnit(equalsC);
}
else
{
```

```
    Serial.println("unknown");
}
if(equation[4]==equalsC)
{
    //motor off
    digitalWrite(8, LOW);
}
else
{
    //motor on
    digitalWrite(8,HIGH);
}
if(calc(equation))
{
    digitalWrite(8, LOW); //motor off
    digitalWrite(10,HIGH);
    digitalWrite(5,LOW);
}
else
{
    digitalWrite(8,HIGH);// motor on
    digitalWrite(5,HIGH);
    digitalWrite(10,LOW);
}
}
}
```