Snake Game

Custom Project Final Report

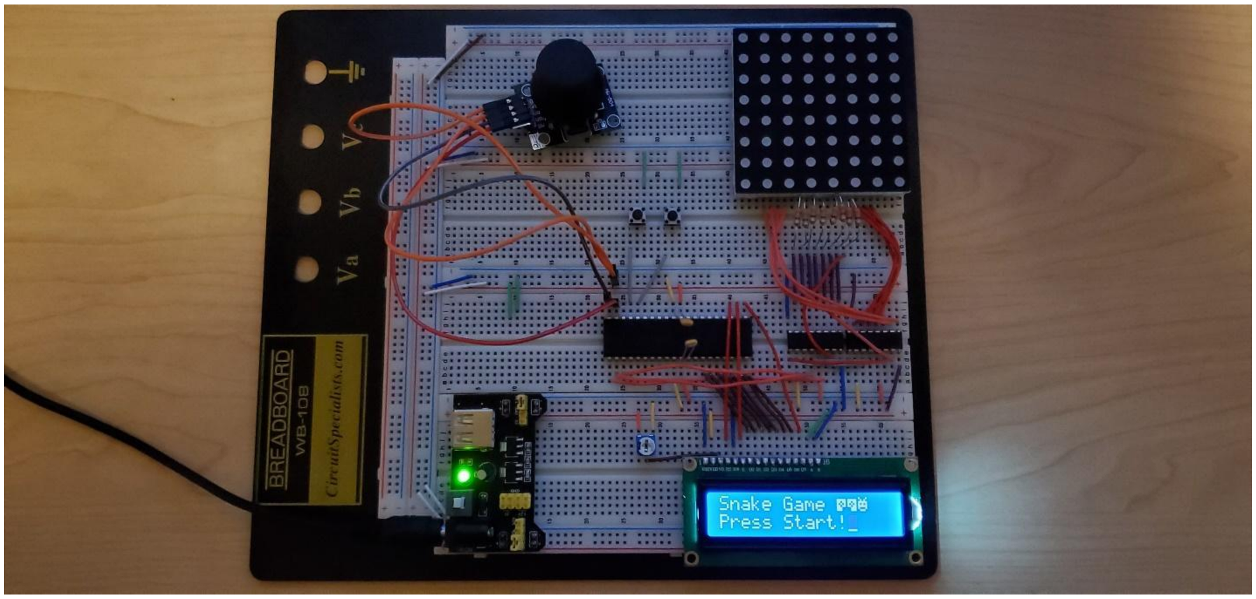
Spring 2019

Anthony Mejia

Table of Contents

Introduction

Snake is a simple game in which the player controls a line (snake) and must collect fruit which are randomly placed on the screen, by moving over it, which increases the length of the line by 1 each time a fruit is consumed. The object of the game is to survive as long as possible and grow as much as you can without colliding with the snake’s own body.



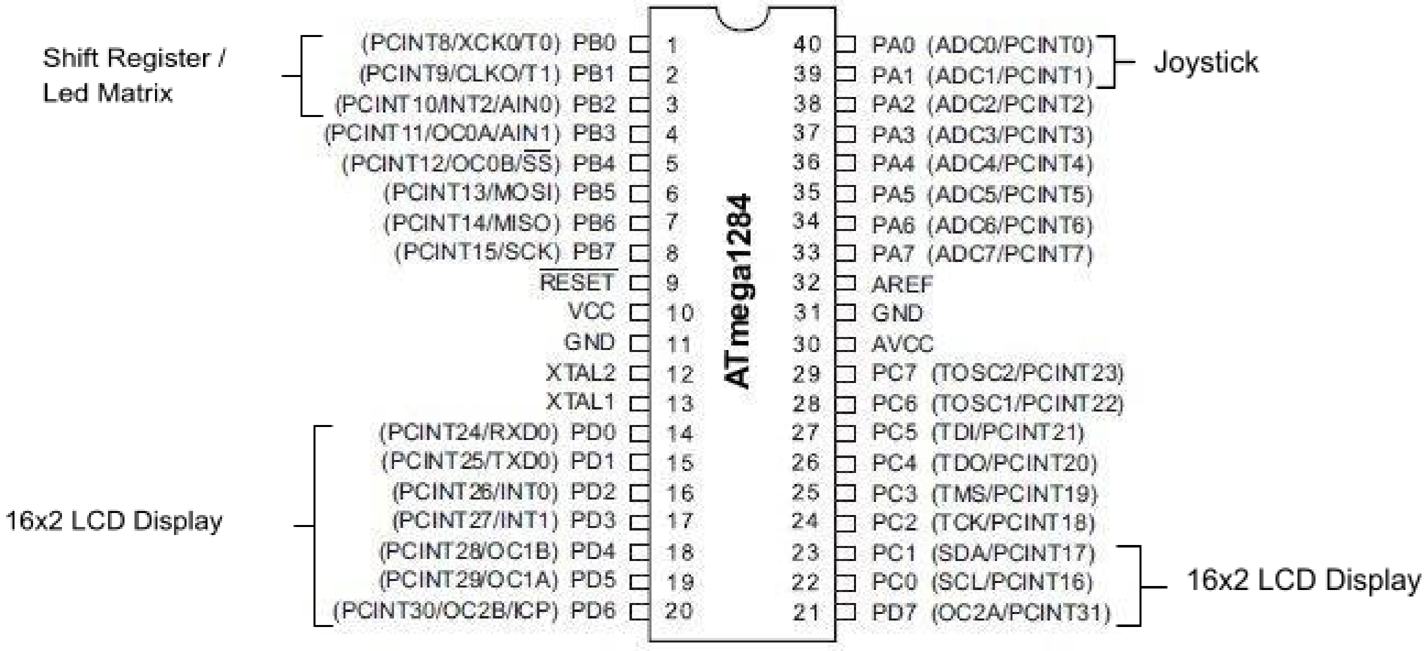
Hardware

Parts List

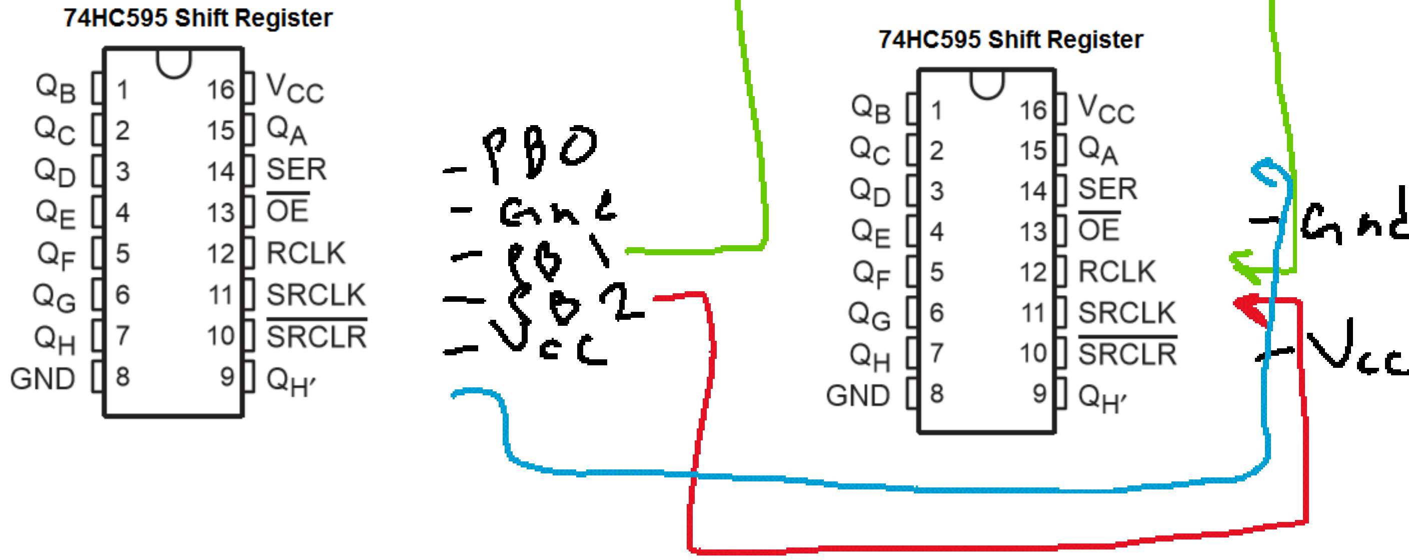
The hardware that was used in this design is listed below. The equipment that was not taught in this course has been bolded.

- ATMega1284 microcontroller
- **8x8 LED Matrix**
- **2 x SN74HC595 Shift Registers**
- **Joystick**
- 16x2 LCD Screen
- 2 x Push Buttons

Pinout



Shift Register / Led Matrix
Joystick
16x2 LCD Display
16x2 LCD Display
ATmega1284
(PCINT8/XCK0/T0) PB0 1
(PCINT9/CLKO/T1) PB1 2
(PCINT10/INT2/AIN0) PB2 3
(PCINT11/OC0A/AIN1) PB3 4
(PCINT12/OC0B/SS) PB4 5
(PCINT13/MOSI) PB5 6
(PCINT14/MISO) PB6 7
(PCINT15/SCK) PB7 8
RESET 9
VCC 10
GND 11
XTAL2 12
XTAL1 13
(PCINT24/RXD0) PD0 14
(PCINT25/TXD0) PD1 15
(PCINT26/INT0) PD2 16
(PCINT27/INT1) PD3 17
(PCINT28/OC1B) PD4 18
(PCINT29/OC1A) PD5 19
(PCINT30/OC2B/ICP) PD6 20
40 PA0 (ADC0/PCINT0)
39 PA1 (ADC1/PCINT1)
38 PA2 (ADC2/PCINT2)
37 PA3 (ADC3/PCINT3)
36 PA4 (ADC4/PCINT4)
35 PA5 (ADC5/PCINT5)
34 PA6 (ADC6/PCINT6)
33 PA7 (ADC7/PCINT7)
32 AREF
31 GND
30 AVCC
29 PC7 (TOSC2/PCINT23)
28 PC6 (TOSC1/PCINT22)
27 PC5 (TDI/PCINT21)
26 PC4 (TDO/PCINT20)
25 PC3 (TMS/PCINT19)
24 PC2 (TCK/PCINT18)
23 PC1 (SDA/PCINT17)
22 PC0 (SCL/PCINT16)
21 PD7 (OC2A/PCINT31)

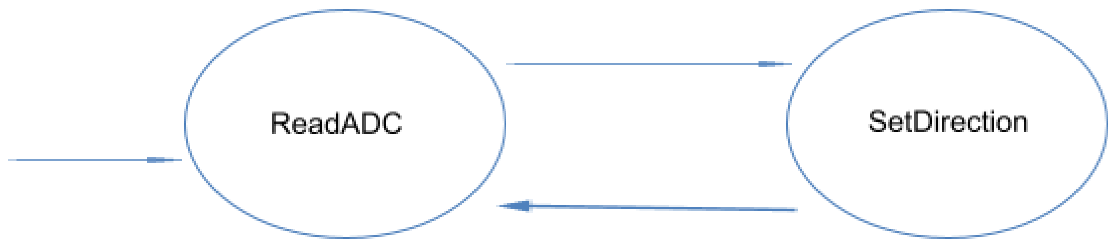


74HC595 Shift Register
QB 1
QC 2
QD 3
QE 4
QF 5
QG 6
QH 7
GND 8
16 VCC
15 QA
14 SER
13 OE
12 RCLK
11 SRCLK
10 SRCLR
9 QH'
74HC595 Shift Register
-PB0
-Gnd
-PB1
-PB2
-Vcc
-Gnd
-Vcc

QA – QH on both registers are connected to the Led Matrix

Software

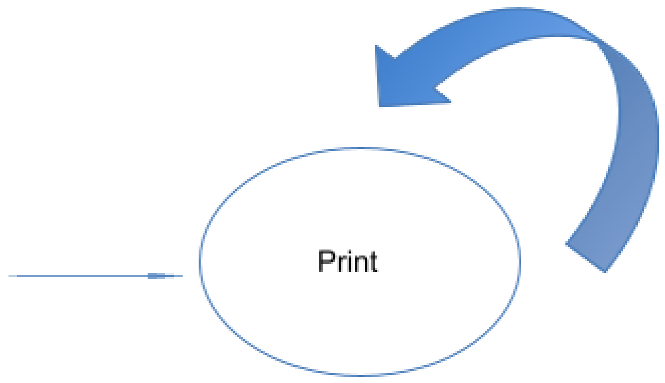
GetADC used to retrieve and set direction based on joystick input.



```
y_direction = adc_read(0);
x_direction = adc_read(1);
```

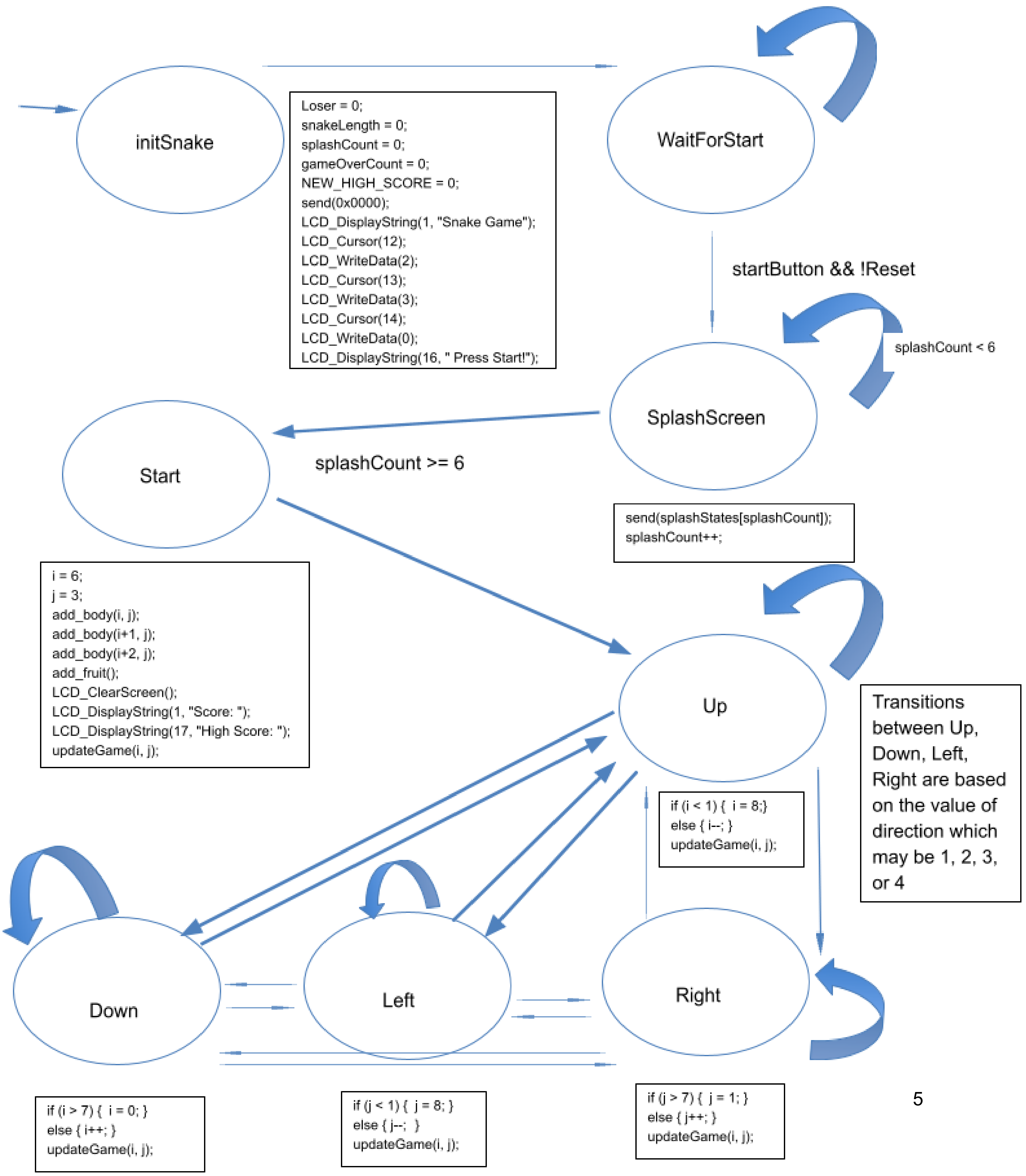
```
if (y_direction >= MAX) { direction=1; }
else if (y_direction <= MIN) {  direction = 2;}
else if (x_direction <= MIN) {direction = 3; }
else if (x_direction >= MAX) { direction = 4;}
else { direction = 0; }
```

PrintSnake used to continuously print the snake on the Led Matrix



```
for (int i = 0; i < snakeLength; i++) {
        uc row = body[i].current.row;
        uc col = body[i].current.col;
        send(getCoordinates(row, col));
        send(getCoordinates(Fruit.row, Fruit.col));
}
```

Snake_Direction used to update the position of the snake on the Led Matrix.



```
Loser = 0;
snakeLength = 0;
splashCount = 0;
gameOverCount = 0;
NEW_HIGH_SCORE = 0;
send(0x0000);
LCD_DisplayString(1, "Snake Game");
LCD_Cursor(12);
LCD_WriteData(2);
LCD_Cursor(13);
LCD_WriteData(3);
LCD_Cursor(14);
LCD_WriteData(0);
LCD_DisplayString(16, " Press Start!");
```

```
send(splashStates[splashCount]);
splashCount++;
```

```
i = 6;
j = 3;
add_body(i, j);
add_body(i+1, j);
add_body(i+2, j);
add_fruit();
LCD_ClearScreen();
LCD_DisplayString(1, "Score: ");
LCD_DisplayString(17, "High Score: ");
updateGame(i, j);
```

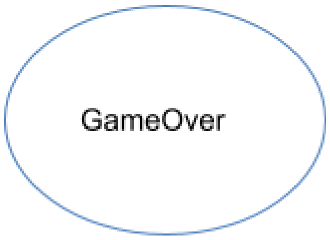
```
if (i < 1) {  i = 8;}
else { i--; }
updateGame(i, j);
```

Transitions between Up, Down, Left, Right are based on the value of direction which may be 1, 2, 3, or 4

```
if (i > 7) {  i = 0; }
else { i++; }
updateGame(i, j);
```

```
if (j < 1) {  j = 8; }
else { j--;  }
updateGame(i, j);
```

```
if (j > 7) {  j = 1; }
else { j++; }
updateGame(i, j);
```



Game over is reached from the Up, Down, Left, Right states when a collision is detected and will then transition back to the initSnake state and clear the screen.

```
if (gameOverCount == 0) {
            LCD_ClearScreen();
LCD_DisplayString(1, "Game Over..");
            if (NEW_HIGH_SCORE) {
                        LCD_DisplayString(17, "HIGH SCORE!!!");
                        LCD_Cursor(31);
                        LCD_WriteData(4);
            }
}
gameOverCount++;
```

Complexities

Completed Complexities:

- Enabling and Writing to the Led Matrix
- Integrating and calibrating the joystick
- Using EEPROM to save the high score
- Creating custom characters on the LCD screen

Incomplete complexities:

No incomplete complexities.

Youtube Link

https://youtu.be/iAvEMSA0AVU

Known Bugs and Shortcomings

- The joystick will occasionally move in the wrong direction. This is either a problem with my threshold values or not filtering the input enough to detect random spikes.
- When resetting the high score during play if the previous score was double digit, the second digit will not be erased. This is most likely caused by me not calling the clear screen function at the appropriate time.

Future work

The next feature I would like to add is a speaker/buzzer for sound effects in order to liven up the game. As well as deal with any bugs left in the system.

References

adc_read function was from http://maxembedded.com/2011/06/the-adc-of-the-avr/

BuildChar function was from https://hackprojects.wordpress.com/forum/avr/display-custom-character-on-16x2-lcd-to-avr-microcontroller-atmega-8/