```
#include <avr/io.h>
#include <stdio.h>
#include <avr/interrupt.h>
#include "io.c"

#define MAX_LEVEL 5

unsigned char victory_level = MAX_LEVEL - 1;

void set_PWM(double frequency) {
	static double current_frequency;
	if (frequency != current_frequency) {
		if (!frequency) { TCCR3B &= 0x08; }
		else { TCCR3B |= 0x03; }
		if (frequency < 0.954) { OCR3A = 0xFFFF; }
		else if (frequency > 31250) { OCR3A = 0x0000; }
		else { OCR3A = (short)(8000000 / (128 * frequency)) - 1; }
		TCNT3 = 0;
		current_frequency = frequency;
	}
	else {}
}

void PWM_on() {
	TCCR3A = (1 << COM3A0);
	TCCR3B = (1 << WGM32) | (1 << CS31) | (1 << CS30);
	set_PWM(0);
}

void PWM_off() {
	TCCR3A = 0x00;
	TCCR3B = 0x00;
}

volatile unsigned char TimerFlag = 0;
unsigned long _avr_timer_M = 1; // Start count from here, down to 0. Default 1 ms.
unsigned long _avr_timer_cntcurr = 0; // Current internal count of 1ms ticks

void TimerOn() {
	// AVR timer/counter controller register TCCR1
	TCCR1B = 0x0B;
	OCR1A = 125;
	TIMSK1 = 0x02;
```

```
	TCNT1=0;

	_avr_timer_cntcurr = _avr_timer_M;

	SREG |= 0x80; // 0x80: 1000000
}

void TimerOff() {
	TCCR1B = 0x00; // bit3bit1bit0=000: timer off
}

void TimerISR() {
	TimerFlag = 1;
}
ISR(TIMER1_COMPA_vect) {
	_avr_timer_cntcurr--; // Count down to 0 rather than up to TOP
	if (_avr_timer_cntcurr == 0) { // results in a more efficient compare
		TimerISR(); // Call the ISR that the user uses
		_avr_timer_cntcurr = _avr_timer_M;
	}
}

void TimerSet(unsigned long M) {
	_avr_timer_M = M;
	_avr_timer_cntcurr = _avr_timer_M;
}


enum Simon_States {init, add_pattern, disp_pattern, Read_pattern, Compare, Level_up, Fail,
Win, disp_pattern_wait, compare_wait,disp_score} state;

unsigned char level = 0;
unsigned char count = 0;
unsigned char pattern[MAX_LEVEL] = {0};
unsigned char freq_counter[9] = {0};
unsigned char tmpA = 0x00, strdA = 0x00;
//unsigned char score = 0;
double freq_test = 0x00;

void SM(){
	//transitions
	tmpA = ~PINA;
```

```
switch (state){
	case init:
	PORTB = 0x00;
	LCD_DisplayString(1, "Welcome");
	if (tmpA){state = add_pattern;
	LCD_DisplayString(1, "Let's Begin");
	}
	else {state = init;}
	level = 0;
	break;

	case add_pattern:

	pattern[level] = rand() % 3;		//Randomising the pattern

	freq_counter[level]  =  pattern[level];  //freq is an array of four freq's... indexable by the pattern[count]... pattern[count] in hex, so its copied to freq_count[count]
	if(pattern[level] == 0)		{pattern[level]= 0x01;}
	else if(pattern[level] == 1)		{pattern[level]= 0x02;}
	else if(pattern[level] == 2)		{pattern[level]= 0x04;}
	else if(pattern[level] == 3)		{pattern[level]= 0x08;}
	//
	state = disp_pattern;
	count = 0;

	case disp_pattern_wait:
	state = disp_pattern;
	PORTB = 0x00;
	break;

	case disp_pattern:
	if(count > level)
	{
		state = Read_pattern;
		PORTB=0x00;
		count = 0;
	}
	else{
		PORTB = pattern[count];
		if(pattern[count]== 0x01)		{set_PWM(2000);}
		else if(pattern[count]== 0x02)		{set_PWM(3000);}
		else if(pattern[count] == 0x04)		{set_PWM(4000);}
		else if(pattern[count] == 0x08)		{set_PWM(5000);}
```

```
        count++;
        state = disp_pattern_wait;
        break;
}


case Read_pattern:
LCD_DisplayString(1, "Press btn");
if(!tmpA)
{
        state = Read_pattern;
        break;
}
else{state = compare_wait;
        strdA = tmpA;
break;}

case compare_wait:
if (!tmpA)
{
        state = Compare;
}
else{
        LCD_DisplayString(1, "Release btn");
        break;}

case Compare:
tmpA = strdA;
if ((count <= level) && (tmpA != pattern[count]))

{
        state = Fail;
}
else if( count < level )                        //had a <=
{
        state = Read_pattern;
        count++;
}
else if (level == victory_level) {state = Win;}
else {state = Level_up;}
break;

case Fail:
```

```
		state = init;
		count = 0;

		////LCD_DisplayString(1, "Wrong, here's the sequence");
		break;

		case Win:
		state = init;
		break;

		case Level_up:
		state = disp_score;
		level++;
		break;

		case disp_score:
		LCD_DisplayString(1, "SCORE : ");
		//score = level ;
		LCD_WriteData(level + '0');
		state = add_pattern;
		break;

		default:
		break;

}
switch (state)
{

		case Fail:
		LCD_DisplayString(1, "FAIL");
		set_PWM(4000);
		PORTB = 0x03;
		break;

		case Level_up:
		LCD_DisplayString(1, "Level up!!!");
		PORTB  =  0x0A;
		set_PWM(2000);
		break;

		case Win:
		LCD_DisplayString(1, "***WIN ***");
```

```
                PORTB = 0x0F;
                set_PWM(3000);
                break;
                default:
                break;
        }


}

int main(void)
{

        DDRA = 0x00; PORTA = 0xFF;
        DDRB = 0xFF; PORTB = 0x00;
        DDRC = 0xFF; PORTC = 0x00; //Confiure port C's 8 pins as ops
        DDRD = 0xFF; PORTD = 0x00;
        LCD_init();
        PWM_on();
        set_PWM(5000);
        TimerSet(100);
        TimerOn();
        state = init;
        //srand (5);
        //RAND_MAX = 3;
        set_PWM(2000);
        while(1)
        {
                //LCD_ClearScreen();
                SM();
                while (!TimerFlag);
                TimerFlag = 0;
                set_PWM(0);
        }       }
```

/*should have seen if !(~PINA) works… ~PINA bring every bit down, ! would have been false the moment one gets high, i.e, is pressed*/