ECE DEPARTMENT

PROJECT NAME : ARDUINO GAME HOUSE

Project done by :ECE SECTION B(B.TECH)

21ECB0B53 - SARVEPALLI MAHATHI

21ECB0B66 - YANDAPALLI HEMANTH KUMAR

21ECB0B63 - VATTAM SAI SHARANYA

Submission To : **Dr. J RAVI KUMAR**

ABSTRACT

Are you bored of playing games in mobile ? And wanted to play some games without using mobile or wanted to create some simple games on your own with your knowledge about Arduino Uno board then this project can help you.
This project has three mini games which we can play in a mobile but the interesting part is we are using microcontroller for playing games and knowing more about Arduino Uno part one of the key features of the project is to combine the fun of video games with the thrill of playing them physically.

<u>COMPONENTS:</u>

1.Arduino UNO board (2 boards)-cost :Rs 1700

2.Ultrasonic Sensor - cost: RS 75

3. SG90 Micro servo motor(5 motors)-cost:Rs 750

4.Joy stick module-cost: Rs 160

5.Jumper pins-cost: Rs 50

6.Bread boards(2 boards) –cost: Rs 120

7.Leds(4)-cost: Rs 8

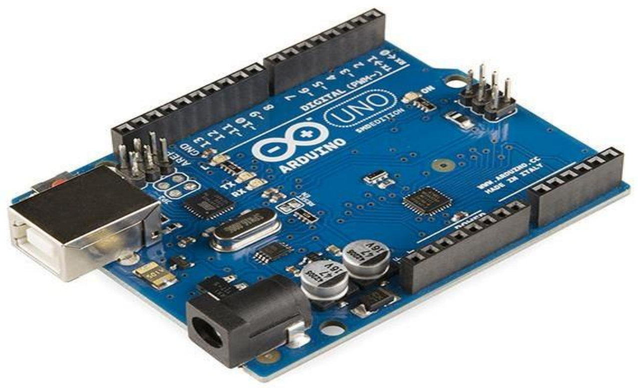
8.Resistors(4 -220 ohms,4-1K ohms)-cost: Rs 8

Required tools:

Cardboard, Scissors, Double tape, Cutting plier.

<u>ABOUT THE COMPONENTS:</u>

<u>ARDUINO BOARD:</u>



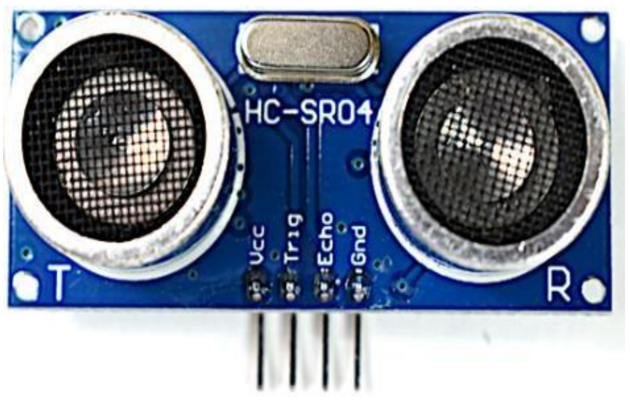
-

Arduino is an open-source electronics platform based on easy-to-use hardware and software Arduino boards are able to read inputs - light on a sensor, a finger on a button, or a Twitter message - and turn it into an output - activating a motor, turning on an LED, publishing something online. You can tell your board what to do by sending a set of instructions to the microcontroller on the board. To do so you use the programming language (based on Wiring), and the Arduino Software (IDE), based on Processing.

Over the years Arduino has been the brain of thousands of projects, from everyday objects to complex scientific instruments. A worldwide community of makers - students, hobbyists, artists, programmers, and professionals - has gathered around this open-source platform, their contributions have added up to an incredible amount of accessible knowledge that can be of great help to novices and experts alike.

Ultrasonic sensor:

An ultrasonic sensor is a proximity sensor that is used to measure the distance of a target or object. It detects the object by transmitting ultrasonic waves and converts the reflected waves into an electrical signal. These sound waves travel faster than the speed of the sound that humans can hear.



It has two main components: the transmitter & receiver. The transmitter emits the sound using a piezoelectric crystal, and the receiver encounters the sound after it has traveled to and from the target. For the calculation of the object distance, the sensor measures the time taken by the signal to travel between the transmission of the sound by the transmitter to the reflecting back towards the receiver.

The formula for this calculation is, $D = \frac{1}{2} T \times C$

Where , D = distance

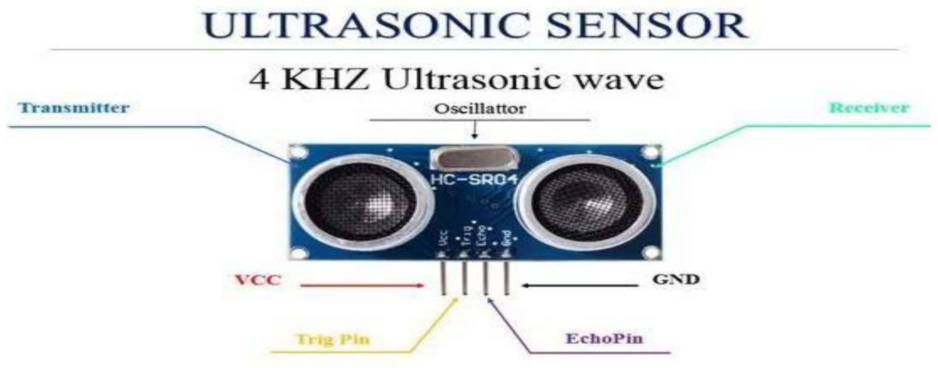
T=time

C=speed of sound (330 m/s)

These sensors are mostly found in automobile self-parking technology and anti-collision safety systems. Also, used in robotic obstacle detection systems, manufacturing technology, and many more. To know more about the ultrasonic sensor, refer to the ultrasonic sensor working principle.

Ultrasonic sensors are used primarily as proximity sensors. They can be found in automobile self-parking technology and anti-collision safety systems.

Ultrasonic sensors are also used in robotic obstacle detection systems, as well as manufacturing technology. In comparison to infrared (IR) sensors in proximity sensing applications, ultrasonic sensors are not as susceptible to interference of smoke, gas, and other airborne particles (though the physical components are still affected by variables such as heat) and are also used as level sensors to detect, monitor, and regulate liquid levels in closed containers (such as vats in chemical factories). Most notably, ultrasonic technology has enabled the medical industry to produce images of internal organs, identify tumors, and ensure the health of babies in the womb.



Micro Servo Motor(SG90):

Micro Servo Motor SG90 is a tiny and lightweight server motor with high output power. Servo can rotate approximately 180 degrees (90 in each direction), and works just like the standard kinds but smaller. You can use any servo code, hardware or library to control these servos. Good for beginners who want to make stuff move without building a motor controller with feedback & gear box, especially since it will fit in small places. It comes with a 3 horns (arms) and hardware.

Micro Servo SG90

Servo motors (or servos) are self-contained electric devices that rotate or push parts of a machine with great precision. Servos are found in many places: from toys to home electronics to cars and airplanes. If you have a radio-controlled model car, airplane, or helicopter, you are using at least a few servos. In a model car or aircraft, servos move levers back and forth to control steering or adjust wing surfaces. By rotating a shaft connected to the engine throttle, a servo regulates the speed of a fuel-powered car or aircraft. Servos also appear behind the scenes in devices we use every day. Electronic devices such as DVD and Blu-ray Disc players use servos to extend or retract the disc trays

The simplicity of a servo is among the features that make them so reliable. The heart of a servo is a small direct current (DC) motor, similar to what you might find in an inexpensive toy. These motors run on electricity from a battery and spin at high RPM (rotations per minute) but put out very low torque . An arrangement of gears takes the high speed of the motor and slows it down while at the same time increasing the torque. T he gear design inside the servo case converts the output to a much slower rotation speed but with more torque (big force, little distance). The amount of actual work is the same, just more useful. Gears in an inexpensive servo motor are generally made of plastic to keep it lighter and less costly.

Motor Connections

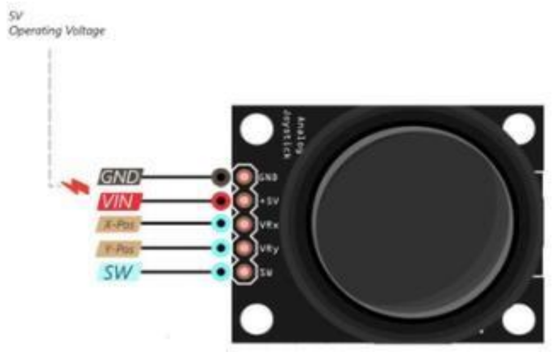
Wire Color	**Connection**
Yellow	PWM
Red	Supply
Brown	Ground

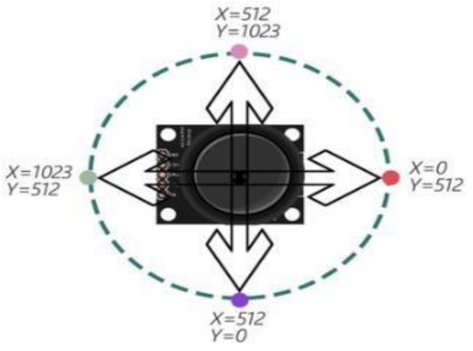
Specifications

General Specifications	
Pulse Width	500µs - 2400µs
Rotation/Support	Bushing
Shaft Diameter	4.5mm
Speed	0.32 oz (9.0 g)
Torque	4.8V: 25.0 oz-in (1.80 kg-cm)
Gear Type	Plastic

General Specifications	
Modulation	Analog
Motor Type	3 Pole Servo Motor
Range	180°
Power Supply	
Phase Voltage	5V

Joystick module:





Joystick module is mostly used in Arduino based DIY projects and Robot Control. The joystick module was used on the transmitter side of this project. It's very easy to connect and fun to use. The module gives analog output. We can use a Joystick Module with Arduino, Raspberry Pi and other Micro controllers. We just need to connect the VRx and VRy axis Pins to the Analog Pins of the microcontroller.

The output range is fixed for all directions. The image above shows the analog output value for the X and Y axis depending on the movement of the Joystick Module in four directions (+X, -X, +Y, -Y). You will also get some analog value as you move the knob diagonally.

Pin Configuration

Pin No.	**Pin Name**	**Description**
1	Gnd	Ground terminal of Module
2	+5v	Positive supply terminal of Module
3	VRx	Voltage Proportional to X axis
4	VRy	Voltage Proportional to Y axis
5	SW	Switch

Features

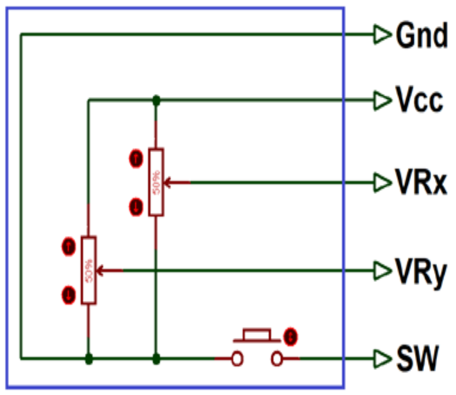
- Two independent Potentiometer: one for each axis (X and Y)
- Auto return to center position
- Low weight
- Cup-type Knob
- Compatible to interface with Arduino or with most microcontrollers

Technical Specifications

- Operating Voltage: 5V
- Internal Potentiometer value: 10k
- 2.54mm pin interface leads
- Dimensions: 1.57 in x 1.02 in x 1.26 in (4.0 cm x 2.6 cm x 3.2 cm)
- Operating temperature: 0 to 70 °C

Internal Structure

The below image is the internal diagram of a Joystick Module. It consists of two Potentiometer, each for one axis (X and Y). Both 10k potentiometer are independent to move in their particular direction. SW (Switch) pin is connected to a push button internally.



DESCRIPTION OF PROJECT:

This project consists of three games:

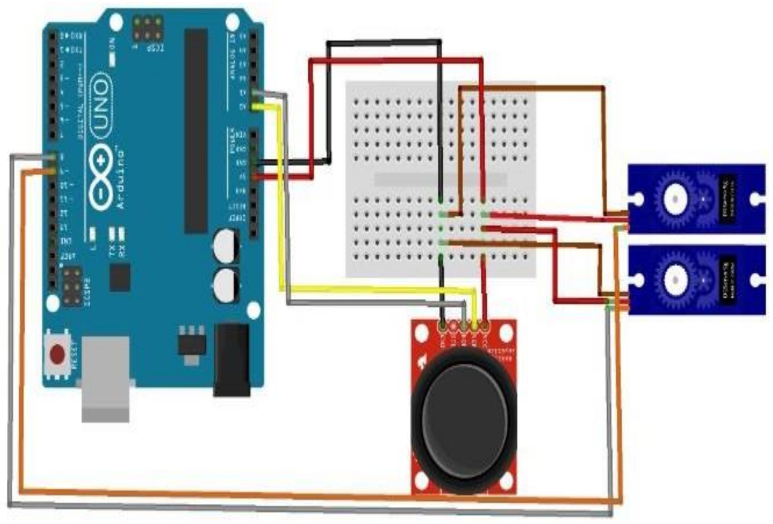
1. Maze game

2. Rock paper scissor

3. Memory game using led bulbs

MAZE GAME:

Circiut diagram:

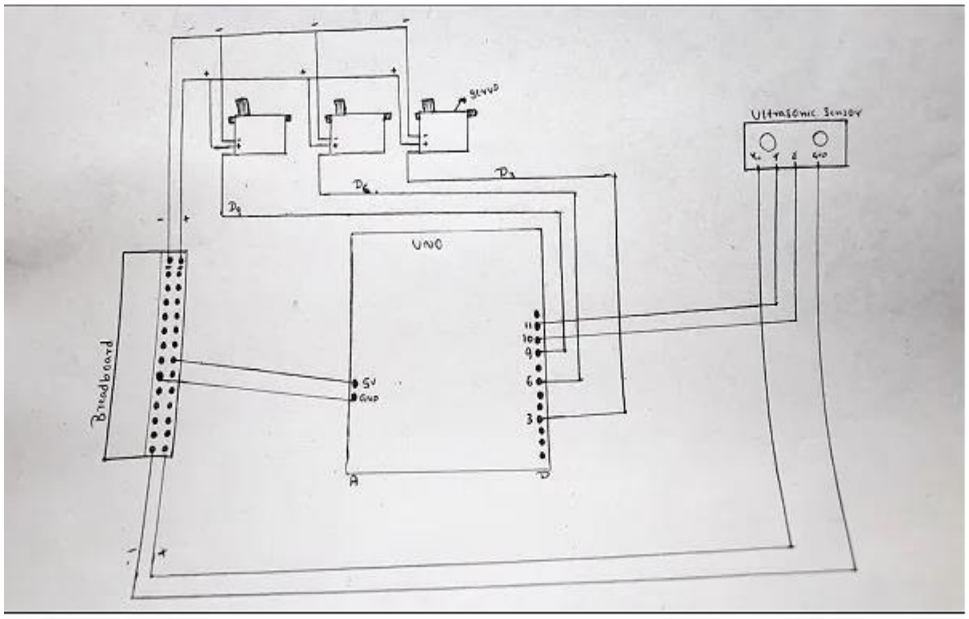


The components used are a joystick module ,two servo motors and a mini bread board. After interfacing the components with arduino board the two servo motor are attached with a double tape in such a way that arm of one motor is attached to body of another then a Maze is attached with the arm of the second motor. The servo motors control the movement of Maze board along x and y directions based on the input to arduino board from joystick module. Without touching the maze physically we can play with it using joystick module by moving it left and right , front and back.

ROCK PAPER SCISSOR:

Rock paper scissor is played by two players but for a change our microcontroller with replace opponent of the player.

Circuit diagram:

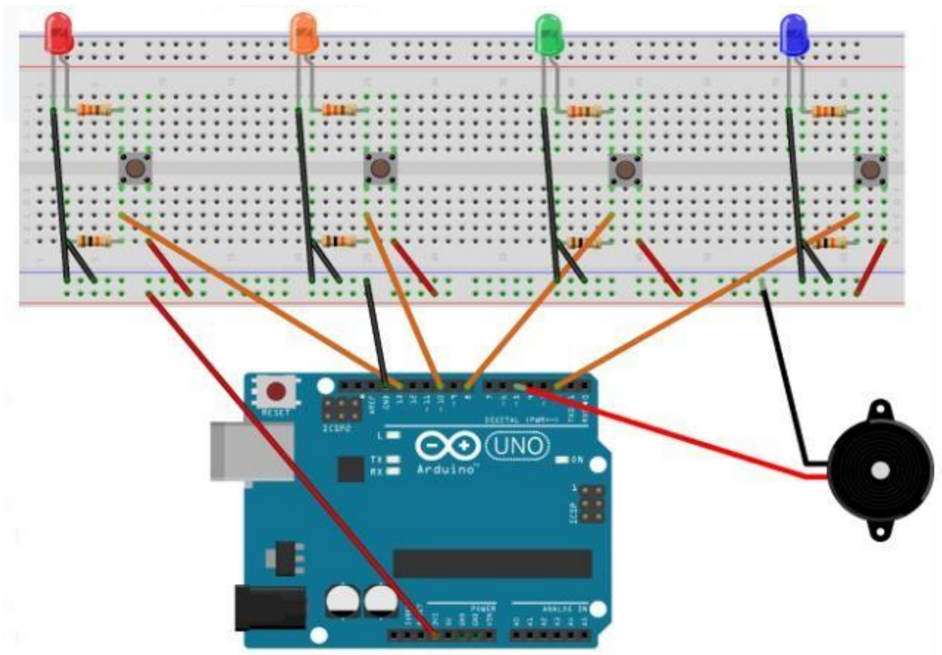


Arduino rock paper scissors circuit diagram

Microcontroller will become the opponent of the player with the help of an ultrasonic sensor and three servo motors .Three servo motors are attached with cardboard pieces having pictures of rock , paper and scissor . Ultrasonic sensor will detect the hands of the player if they are in the range of it and gives input to arduino board which will give output to any of the servomotors (the servo motor is randomly selected).

MEMORY GAME USING LEDS

Circuit diagram:

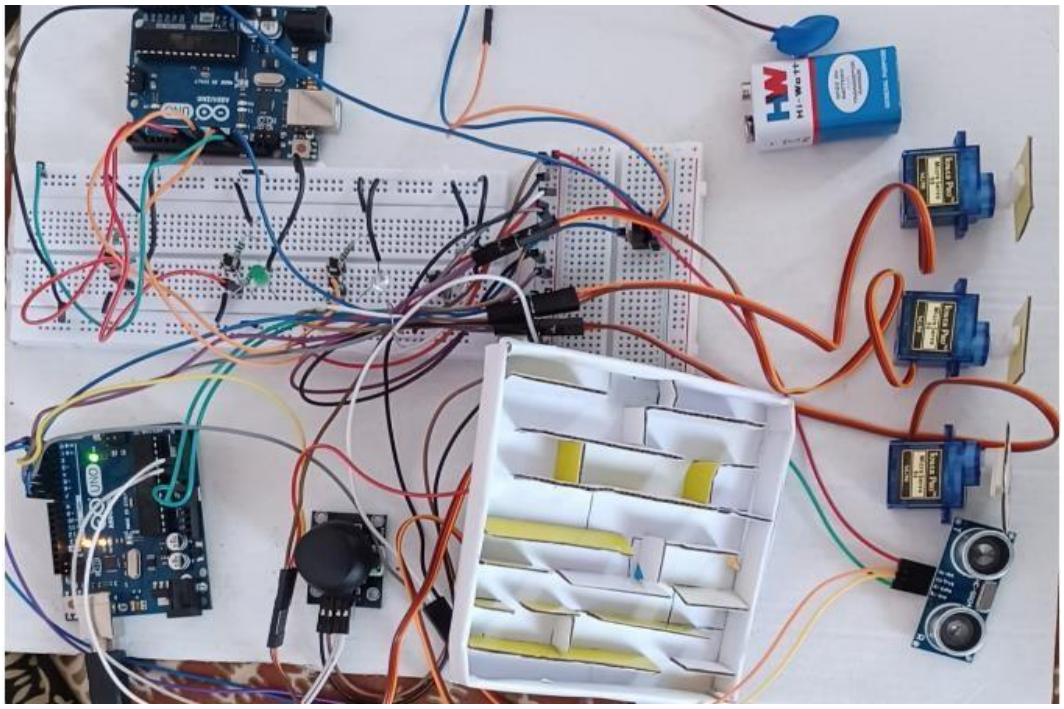


The game circuit uses 4 LEDs and 4 push buttons. As soon as the game turns on, all the LEDs blink 4 times and then the game begins.

The Arduino blinks a random led first, then the user is required to immediately press the corresponding button, if the button is correct then the Arduino blinks 2 leds randomly, and this continues until the user presses the buttons in the wrong sequence.

Once the user presses the wrong sequence, all the leds blink at high speed, then the right sequence is shown, after which the game restarts.

Complete Circuit diagram:



AURDINO CODES:

we used two arduino boards first board is for Maze and rock paper scissor games and second one is for Memory game.

Code 1:

```
#include<Servo.h>

Servo sx;

Servo sy;//defining our servos

//defining joystick pins

int joyx=A0;

int joyy=A1;

int joyval;//variable to read value from analog pins
```

```
volatile long A;



float checkdistance_0_1()

{

digitalWrite(0, LOW);

delayMicroseconds(2);

digitalWrite(0, HIGH);

delayMicroseconds(10);

digitalWrite(0, LOW);

float distance = pulseIn(1, HIGH) / 58.00;

delay(10);

return distance;

}



Servo servo_7;

Servo servo_4;

Servo servo_6;
```

```
void setup()

{

  sx.attach(3);//attaching digital pins

  sy.attach(5);

  pinMode(0, OUTPUT);

pinMode(1, INPUT);

servo_7.attach(7);    // stone

servo_4.attach(4);    // paper

servo_6.attach(6);     // scissor

servo_7.write(179);

servo_4.write(179);

servo_6.write(179);

}


void loop()

{

  //read the value of joystick (between 0 to 1023)

  joyval=analogRead(joyx);
  joyval=map(joyval,0,1023,0,180);

  sx.write(joyval);//set the servo position according to the joystick value

  joyval=analogRead(joyy);

  joyval=map(joyval,0,1023,0,180);

  sy.write(joyval);

  delay(15);

  if (checkdistance_0_1() < 10) {

A = random(0, 4);

switch (A) {

case 1:



servo_7.write(120);



delay(1000);

servo_7.write(179);



delay(500);

break;
```

```
case 2:


servo_4.write(120);


delay(1000);

servo_4.write(179);


delay(500);

break;

case 3:


servo_6.write(120);


delay(1000);

servo_6.write(179);


delay(500);

break;
```

```
}

}

}
```

Code 2:

```
#define PLAYER_WAIT_TIME 3000 // The time allowed between button presses -
3s



byte sequence[100];          // Storage for the light sequence

byte curLen = 0;             // Current length of the sequence

byte inputCount = 0;         // The number of times that the player has pressed a
(correct) button in a given turn

byte lastInput = 0;          // Last input from the player

byte expRd = 0;              // The LED that's suppose to be lit by the player

bool btnDwn = false;         // Used to check if a button is pressed

bool wait = false;           // Is the program waiting for the user to press a button

bool resetFlag = false;      // Used to indicate to the program that once the player
lost



//byte soundPin = 5;           // Speaker output
```

```
byte noPins = 4;              // Number of buttons/LEDs (While working on this, I was
using only 2 LEDs)

                              // You could make the game harder by adding an additional
LED/button/resistors combination.

byte pins[] = {2, 13, 10, 8}; // Button input pins and LED ouput pins - change these
vaules if you wwant to connect yourbuttons to other pins

                              // The number of elements must match noPins below



long inputTime = 0;           // Timer variable for the delay between user inputs



void setup() {

  delay(3000);                // This is to give me time to breathe after connection the
arduino - can be removed if you want

  Serial.begin(9600);         // Start Serial monitor. This can be removed too as long
as you remove all references to Serial below

  Reset();

}



///

/// Sets all the pins as either INPUT or OUTPUT based on the value of 'dir'
```

```
///

void setPinDirection(byte dir){

  for(byte i = 0; i < noPins; i++){

  pinMode(pins[i], dir);

  }

}


//send the same value to all the LED pins

void writeAllPins(byte val){

  for(byte i = 0; i < noPins; i++){

    digitalWrite(pins[i], val);

  }

}


//Makes a (very annoying :) beep sound

//void beep(byte freq){

  //analogWrite(soundPin, 2);

  //delay(freq);
  //analogWrite(soundPin, 0);

  //delay(freq);

//}



///

/// Flashes all the LEDs together

/// freq is the blink speed - small number -> fast | big number -> slow

///

void flash(short freq){

  setPinDirection(OUTPUT); /// We're activating the LEDS now

  for(int i = 0; i < 5; i++){

    writeAllPins(HIGH);

  //  beep(50);

    delay(freq);

    writeAllPins(LOW);

    delay(freq);

  }

}
```

```
///

///This function resets all the game variables to their default values

///

void Reset(){

  flash(500);

  curLen = 0;

  inputCount = 0;

  lastInput = 0;

  expRd = 0;

  btnDwn = false;

  wait = false;

  resetFlag = false;

}


///

/// User lost

///
```

```
void Lose(){

  flash(50);

}



///

/// The arduino shows the user what must be memorized

/// Also called after losing to show you what you last sequence was

///

void playSequence(){

  //Loop through the stored sequence and light the appropriate LEDs in turn

  for(int i = 0; i < curLen; i++){

      Serial.print("Seq: ");

      Serial.print(i);

      Serial.print("Pin: ");

      Serial.println(sequence[i]);

      digitalWrite(sequence[i], HIGH);

      delay(500);

      digitalWrite(sequence[i], LOW);
      delay(250);

    }

}



///

/// The events that occur upon a loss

///

void DoLoseProcess(){

  Lose();             // Flash all the LEDS quickly (see Lose function)

  delay(1000);

  playSequence();     // Shows the user the last sequence - So you can count
remember your best score - Mine's 22 by the way :)

  delay(1000);

  Reset();            // Reset everything for a new game

}



///

/// Where the magic happens

///
```

```
void loop() {

  if(!wait){

                      //******//

                      // Arduino's turn //

                      //******//

    setPinDirection(OUTPUT);                    // We're using the LEDs



    randomSeed(analogRead(A0));                 //
https://www.arduino.cc/en/Reference/RandomSeed

    sequence[curLen] = pins[random(0,noPins)];    // Put a new random value in the
next position in the sequence -  https://www.arduino.cc/en/Reference/random

    curLen++;                              // Set the new Current length of the sequence



    playSequence();                          // Show the sequence to the player

  //  beep(50);                             // Make a beep for the player to be aware



    wait = true;                           // Set Wait to true as it's now going to be the
turn of the player

    inputTime = millis();                   // Store the time to measure the player's
response time
```

```
  }else{

                    //*****//

                    // Player's turn //

                    //*****//

    setPinDirection(INPUT);                    // We're using the buttons



    if(millis() - inputTime > PLAYER_WAIT_TIME){  // If the player takes more than
the allowed time,

      DoLoseProcess();                         // All is lost :(

      return;

    }



    if(!btnDwn){                               //

      expRd = sequence[inputCount];            // Find the value we expect from the
player

      Serial.print("Expected: ");              // Serial Monitor Output - Should be
removed if you removed the Serial.begin above

      Serial.println(expRd);                   // Serial Monitor Output - Should be
removed if you removed the Serial.begin above
```

```
   for(int i = 0; i < noPins; i++){          // Loop through the all the pins

    if(pins[i]==expRd)

     continue;                          // Ignore the correct pin

    if(digitalRead(pins[i]) == HIGH){        // Is the buttong pressed

    lastInput = pins[i];

     resetFlag = true;                    // Set the resetFlag - this means you lost

     btnDwn = true;                        // This will prevent the program from doing
the same thing over and over again

     Serial.print("Read: ");               // Serial Monitor Output - Should be removed
if you removed the Serial.begin above

     Serial.println(lastInput);             // Serial Monitor Output - Should be
removed if you removed the Serial.begin above

    }

   }

  }



  if(digitalRead(expRd) == 1 && !btnDwn)        // The player pressed the right
button

  {

   inputTime = millis();                     //
```

```
      lastInput = expRd;

      inputCount++;                            // The user pressed a (correct) button again

      btnDwn = true;                           // This will prevent the program from doing
the same thing over and over again

      Serial.print("Read: ");                  // Serial Monitor Output - Should be removed
if you removed the Serial.begin above

      Serial.println(lastInput);               // Serial Monitor Output - Should be
removed if you removed the Serial.begin above

    }else{

      if(btnDwn && digitalRead(lastInput) == LOW){  // Check if the player released
the button

        btnDwn = false;

        delay(20);

        if(resetFlag){                         // If this was set to true up above, you lost

          DoLoseProcess();                     // So we do the losing sequence of events

        }

        else{

          if(inputCount == curLen){            // Has the player finished repeating the
sequence

            wait = false;                      // If so, this will make the next turn the
program's turn
```

```
          inputCount = 0;                      // Reset the number of times that the player
has pressed a button

          delay(1500);

        }

      }

    }

   }

 }

}
```