

TEST CODE

```
#define MUTE 18
#define HOLE01 1
#define HOLE02 19
#define HOLE1 10
#define HOLE2 4
#define HOLE3 6
#define HOLE4 5
#define HOLE5 7
#define HOLE6 20
#define HOLE7 21

#define freq 50
#define resolution 12
#define PWMCHANNEL0 0
#define PWMCHANNEL1 1
#define PWMCHANNEL2 2
#define PWMCHANNEL3 3
#define PWMCHANNEL4 4
#define PWMCHANNEL5 5

#define closed01 205
#define open01 220
#define closed02 105
#define open02 120
#define closed2 215
#define open2 200
#define closed3 235
#define open3 230
#define closed4 235
#define open4 200
#define closed5 265
#define open5 240

int min_counts = 102;
int max_counts = 501;

struct ServoState {
    int pin;
    float currentPos;
    float targetPos;
    float speed;
};
```

```

struct Pitch {
    int hole01;
    int hole02;
    int hole1;
    int hole2;
    int hole3;
    int hole4;
    int hole5;
    int hole6;
    int hole7;
};

int Midi_Note[] = {
    84, 86, 88, 89, 91, 93, 95, 96, 98, 100
};

int start_time[] = {
    1000, 2000, 3000, 4000, 5000, 6000, 7000, 8000, 9000 ,10000
};

int end_time[] = {
    2000, 3000, 4000, 5000, 6000, 7000, 8000, 9000 ,10000, 11000
};

const Pitch NOTE_A5 {closed01,closed02,0,closed2,open3,open4,open5,0,0}; // MIDI 81,
24V
const Pitch NOTE_B5 {closed01,closed02,0,open2,open3,open4,open5,0,0}; //MIDI 83,24V
const Pitch NOTE_C6 {closed01,closed02,0,closed2,closed3,closed4,closed5,1,1}; //
MIDI84,24V
const Pitch NOTE_D6 {closed01,closed02,0,closed2,closed3,closed4,closed5,1,0}; //
MIDI86,24V
const Pitch NOTE_E6 {closed01,closed02,0,closed2,closed3,closed4,closed5,0,0}; //
MIDI88,24V
const Pitch NOTE_F6 {closed01,closed02,0,closed2,closed3,closed4,open5,0,0}; //
MIDI89,24V
const Pitch NOTE_G6 {closed01,closed02,0,closed2,closed3,open4,open5,0,0}; //
MIDI91,24V
const Pitch NOTE_A6 {closed01,closed02,0,closed2,open3,closed4,closed5,1,0}; //
MIDI93,24V
const Pitch NONE {open01,open02,0,open2,open3,open4,open5,0,0}; // 其余音符全部静音

```

```

const int NOTE_COUNT = sizeof(Midi_Note) / sizeof(Midi_Note[0]);

uint32_t timebase;
int currentNote = 0;
bool noteActive = false;

ServoState Servos[6];
unsigned long lastServoUpdate = 0;

void moveServo(int servoNumber, int target, int movementTime) {
    int targetClamped = constrain(target, min_counts, max_counts);
    Servos[servoNumber].targetPos = targetClamped;

    float distance = abs(targetClamped - Servos[servoNumber].currentPos);
    int updateTimes = movementTime / 10;
    Servos[servoNumber].speed = distance / updateTimes;

    if (Servos[servoNumber].speed < 0.2)
        Servos[servoNumber].speed = 0.2;
}

void updateServo(ServoState &servo) {
    if (servo.currentPos == servo.targetPos) return;

    if (servo.currentPos < servo.targetPos) {
        servo.currentPos += servo.speed;
        if (servo.currentPos > servo.targetPos) servo.currentPos = servo.targetPos;
    } else {
        servo.currentPos -= servo.speed;
        if (servo.currentPos < servo.targetPos) servo.currentPos = servo.targetPos;
    }
    ledcWrite(servo.pin, (int)servo.currentPos);
}

Pitch midiToPitch(int midiNum) {
    switch(midiNum) {
        // case 72: return NOTE_C5;
        // case 74: return NOTE_D5;
        // case 76: return NOTE_E5;
        // case 77: return NOTE_F5;
    }
}

```

```

// case 79: return NOTE_G5;

case 81: return NOTE_A5;
case 83: return NOTE_B5;
case 84: return NOTE_C6;
case 86: return NOTE_D6;
case 88: return NOTE_E6;
case 89: return NOTE_F6;
case 91: return NOTE_G6;
case 93: return NOTE_A6;

default: return NONE;
}
}

void playMidiSong() {
    unsigned long songTime = millis() - timebase;

    if (noteActive && currentNote > 0) {
        const NoteEvent *prev = midi.getNote(currentNote - 1);
        if (songTime >= prev->endTimeMs) {
            Pitch mutePitch = NONE;
            moveServo(0, mutePitch.hole01, 150);
            moveServo(1, mutePitch.hole02, 150);
            moveServo(2, mutePitch.hole2, 150);
            moveServo(3, mutePitch.hole3, 150);
            moveServo(4, mutePitch.hole4, 150);
            moveServo(5, mutePitch.hole5, 150);

            digitalWrite(HOLE1, mutePitch.hole1);
            digitalWrite(HOLE6, mutePitch.hole6);
            digitalWrite(HOLE7, mutePitch.hole7);
            noteActive = false;
        }
    }
    if (currentNote < midi.getNoteCount()) {
        const NoteEvent *note = midi.getNote(currentNote);
        if (songTime >= note->startTimeMs) {
            Pitch curPitch = midiToPitch(note->pitch);

            moveServo(0, curPitch.hole01, 150);
            moveServo(1, curPitch.hole02, 150);

```

```

        moveServo(2, curPitch.hole2, 150);
        moveServo(3, curPitch.hole3, 150);
        moveServo(4, curPitch.hole4, 150);
        moveServo(5, curPitch.hole5, 150);

        digitalWrite(HOLE1, curPitch.hole1);
        digitalWrite(HOLE6, curPitch.hole6);
        digitalWrite(HOLE7, curPitch.hole7);

        noteActive = true;
        currentNote++;
    }
}

if (currentNote >= midi.getNoteCount() && !noteActive) {
    currentNote = 0;
    timebase = millis();
}
}

void setup() {
    delay(10000);
    Serial.begin(921600);

    ledcAttachChannel(HOLE01, freq, resolution, PWMCHANNEL0);
    ledcAttachChannel(HOLE02, freq, resolution, PWMCHANNEL1);
    ledcAttachChannel(HOLE2, freq, resolution, PWMCHANNEL2);
    ledcAttachChannel(HOLE3, freq, resolution, PWMCHANNEL3);
    ledcAttachChannel(HOLE4, freq, resolution, PWMCHANNEL4);
    ledcAttachChannel(HOLE5, freq, resolution, PWMCHANNEL5);

    ledcWrite(HOLE01, open01);
    ledcWrite(HOLE02, open02);
    ledcWrite(HOLE2, open2);
    ledcWrite(HOLE3, open3);
    ledcWrite(HOLE4, open4);
    ledcWrite(HOLE5, open5);
    digitalWrite(HOLE1, 0);
    digitalWrite(HOLE6, 0);
    digitalWrite(HOLE7, 0);
    digitalWrite(MUTE, 0);
}

```

```
Servos[0] = {HOLE01, open01, open01, 1.0};
Servos[1] = {HOLE02, open02, open02, 1.0};
Servos[2] = {HOLE2, open2, open2, 1.0};
Servos[3] = {HOLE3, open3, open3, 1.0};
Servos[4] = {HOLE4, open4, open4, 1.0};
Servos[5] = {HOLE5, open5, open5, 1.0};

pinMode(HOLE1, OUTPUT);
pinMode(HOLE6, OUTPUT);
pinMode(HOLE7, OUTPUT);
pinMode(MUTE, OUTPUT);

delay(1000);
Serial.printf("MIDI音符总数: %d\n", NOTE_COUNT);
Serial.println("演奏开始!");

//midi.fetchByTrackName("Song.mid", "Recorder");
timebase = millis();
}

void loop() {
    if (millis() - lastServoUpdate >= 10) {
        updateServo(Servos[0]);
        updateServo(Servos[1]);
        updateServo(Servos[2]);
        updateServo(Servos[3]);
        updateServo(Servos[4]);
        updateServo(Servos[5]);
        lastServoUpdate = millis();
    }

    playMidiSong();
}
```

CODE w/ WIFI

CODE w/ WIFI

```
#include <WiFi.h>
#include <MidiData.h>

#define MUTE 18
#define HOLE01 1
#define HOLE02 19
#define HOLE1 10
#define HOLE2 4
#define HOLE3 6
#define HOLE4 5
#define HOLE5 7
#define HOLE6 20
#define HOLE7 21

#define freq 50
#define resolution 12
#define PWMCHANNEL0 0
#define PWMCHANNEL1 1
#define PWMCHANNEL2 2
#define PWMCHANNEL3 3
#define PWMCHANNEL4 4
#define PWMCHANNEL5 5

#define closed01 205
#define open01 220
#define closed02 105
#define open02 120
#define closed2 215
#define open2 200
#define closed3 235
#define open3 230
#define closed4 235
#define open4 200
#define closed5 265
#define open5 240

int min_counts = 102;
int max_counts = 501;
```

```

struct ServoState {
    int pin;
    float currentPos;
    float targetPos;
    float speed;
};

struct Pitch {
    int hole01;
    int hole02;
    int hole1;
    int hole2;
    int hole3;
    int hole4;
    int hole5;
    int hole6;
    int hole7;
};

const Pitch NOTE_G5 {closed01,closed02,0,closed2,closed3,open4,open5,0,0}; //79, 13V
const Pitch NOTE_A5 {closed01,closed02,0,closed2,open3,open4,open5,0,0}; // MIDI 81,
24V
const Pitch NOTE_B5 {closed01,closed02,0,open2,open3,open4,open5,0,0}; //MIDI 83,24V
const Pitch NOTE_C6 {closed01,closed02,0,closed2,closed3,closed4,closed5,1,1}; //
MIDI84,24V
const Pitch NOTE_D6 {closed01,closed02,0,closed2,closed3,closed4,closed5,1,0}; //
MIDI86,24V
const Pitch NOTE_E6 {closed01,closed02,0,closed2,closed3,closed4,closed5,0,0}; //
MIDI88,24V
const Pitch NOTE_F6 {closed01,closed02,0,closed2,closed3,closed4,open5,0,0}; //
MIDI89,24V
const Pitch NOTE_G6 {closed01,closed02,0,closed2,closed3,open4,open5,0,0}; //
MIDI91,24V
const Pitch NOTE_A6 {closed01,closed02,0,closed2,open3,closed4,closed5,1,0}; //
MIDI93,24V
const Pitch NONE {open01,open02,0,open2,open3,open4,open5,0,0}; // 其余音符全部静音

int minRange = 81;
int maxRange = 93;

// int Midi_Note[] = {
//     64, 72, 72, 72, 64, 64, 72, 72, 72, 69,

```

```
// 69, 60, 60, 65, 77, 69, 69, 69, 81, 74,
// 81, 71, 69, 71, 72, 71, 69, 67, 64, 77,
// 76, 74, 72, 74, 76, 74, 72, 71, 71, 69,
// 71, 72, 71, 69, 67, 64, 77, 76, 74, 72,
// 76, 79, 72, 72, 71, 69, 65, 72, 72, 71,
// 69, 65
// };

// int start_time[] = {
// 0,357,5357,6786,7143,
// 7500,8214,9643,10357,11071,
// 12500,12857,13214,13571,13929,
// 14643,14821,15000,15357,15714,
// 16071,27857,30357,30536,30714,
// 30893,31071,31250,31429,33393,
// 34643,34821,35000,35893,36071,
// 36250,37500,38571,39286,41786,
// 41964,42143,42321,42500,42679,
// 42857,44821,46071,46250,46429,
// 47500,47857,56429,60714,61429,
// 62143,65000,67857,72143,72857,
// 73571,76429
// };

// int end_time[] = {
// 3392,3410,5696,7124,7481,
// 7678,8553,9981,10535,11410,
// 12839,13196,13392,13910,14267,
// 14731,14910,15339,15696,16053,
// 16249,30231,30526,30705,30883,
// 31062,31240,31419,32106,34580,
// 34812,34990,35763,36062,36240,
// 37437,38517,39249,41660,41955,
// 42133,42312,42490,42669,42847,
// 43535,46008,46240,46419,47446,
// 47839,49553,60499,61392,62106,
// 64856,67714,71928,72821,73535,
// 76285,79142
// };

const int NOTE_COUNT = sizeof(Midi_Note) / sizeof(Midi_Note[0]);
```

```

uint32_t timebase;
int currentNote = 0;
bool noteActive = false;
MidiData midi("192.168.8.173", 5000);

ServoState Servos[6];
unsigned long lastServoUpdate = 0;

void moveServo(int servoNumber, int target, int movementTime) {
    int targetClamped = constrain(target, min_counts, max_counts);
    Servos[servoNumber].targetPos = targetClamped;

    float distance = abs(targetClamped - Servos[servoNumber].currentPos);
    int updateTimes = movementTime / 10;
    Servos[servoNumber].speed = distance / updateTimes;

    if (Servos[servoNumber].speed < 0.2)
        Servos[servoNumber].speed = 0.2;
}

void updateServo(ServoState &servo) {
    if (servo.currentPos == servo.targetPos) return;

    if (servo.currentPos < servo.targetPos) {
        servo.currentPos += servo.speed;
        if (servo.currentPos > servo.targetPos) servo.currentPos = servo.targetPos;
    } else {
        servo.currentPos -= servo.speed;
        if (servo.currentPos < servo.targetPos) servo.currentPos = servo.targetPos;
    }

    ledcWrite(servo.pin, (int)servo.currentPos);
}

Pitch midiToPitch(int midiNum) {
    switch(midiNum) {
        // case 72: return NOTE_C5;
        // case 74: return NOTE_D5;
        // case 76: return NOTE_E5;
        // case 77: return NOTE_F5;
        // case 79: return NOTE_G5;

        case 81: return NOTE_A5;
    }
}

```

```

    case 83: return NOTE_B5;
    case 84: return NOTE_C6;
    case 86: return NOTE_D6;
    case 88: return NOTE_E6;
    case 89: return NOTE_F6;
    case 91: return NOTE_G6;
    case 93: return NOTE_A6;

    default: return NONE;
}
}

void playMidiSong() {
    unsigned long songTime = millis() - timebase;
    if (noteActive && currentNote > 0) {
        const NoteEvent *prev = midi.getNote(currentNote - 1);
        if (songTime >= prev->endTimeMs) {
            Pitch mutePitch = NONE;

            moveServo(0, mutePitch.hole01, 150);
            moveServo(1, mutePitch.hole02, 150);
            moveServo(2, mutePitch.hole2, 150);
            moveServo(3, mutePitch.hole3, 150);
            moveServo(4, mutePitch.hole4, 150);
            moveServo(5, mutePitch.hole5, 150);

            digitalWrite(HOLE1, mutePitch.hole1);
            digitalWrite(HOLE6, mutePitch.hole6);
            digitalWrite(HOLE7, mutePitch.hole7);
            noteActive = false;
        }
    }
    if (currentNote < midi.getNoteCount()) {
        const NoteEvent *note = midi.getNote(currentNote);
        if (songTime >= note->startTimeMs) {
            Pitch curPitch = midiToPitch(note->pitch);

            moveServo(0, curPitch.hole01, 150);
            moveServo(1, curPitch.hole02, 150);
            moveServo(2, curPitch.hole2, 150);
            moveServo(3, curPitch.hole3, 150);
            moveServo(4, curPitch.hole4, 150);
            moveServo(5, curPitch.hole5, 150);

```

```

        digitalWrite(HOLE1, curPitch.hole1);
        digitalWrite(HOLE6, curPitch.hole6);
        digitalWrite(HOLE7, curPitch.hole7);

        noteActive = true;
        currentNote++;
    }
}

if (currentNote >= midi.getNoteCount() && !noteActive) {
    currentNote = 0;
    timebase = millis();
}
}

//MidiData midi("192.168.8.173", 5000);
void setup() {
    delay(10000);
    Serial.begin(921600);

    WiFi.begin(" ", " ");

    while (WiFi.status() != WL_CONNECTED) {
        Serial.print(".");
        delay(500);
    }

    Serial.println();
    Serial.println("WiFi Connected");
    midi.fetchByTrackName("Song.mid", "Recorder");
    Serial.printf("Downloaded %d notes\n", midi.getNoteCount());

    ledcAttachChannel(HOLE01, freq, resolution, PWMCHANNEL0);
    ledcAttachChannel(HOLE02, freq, resolution, PWMCHANNEL1);
    ledcAttachChannel(HOLE2, freq, resolution, PWMCHANNEL2);
    ledcAttachChannel(HOLE3, freq, resolution, PWMCHANNEL3);
    ledcAttachChannel(HOLE4, freq, resolution, PWMCHANNEL4);
    ledcAttachChannel(HOLE5, freq, resolution, PWMCHANNEL5);

    ledcWrite(HOLE01, open01);
    ledcWrite(HOLE02, open02);

```

```

ledcWrite(HOLE2, open2);
ledcWrite(HOLE3, open3);
ledcWrite(HOLE4, open4);
ledcWrite(HOLE5, open5);
digitalWrite(HOLE1,0);
digitalWrite(HOLE6,0);
digitalWrite(HOLE7,0);
digitalWrite(MUTE,0);

Servos[0] = {HOLE01, open01, open01, 1.0};
Servos[1] = {HOLE02, open02, open02, 1.0};
Servos[2] = {HOLE2, open2, open2, 1.0};
Servos[3] = {HOLE3, open3, open3, 1.0};
Servos[4] = {HOLE4, open4, open4, 1.0};
Servos[5] = {HOLE5, open5, open5, 1.0};

pinMode(HOLE1, OUTPUT);
pinMode(HOLE6, OUTPUT);
pinMode(HOLE7, OUTPUT);
pinMode(MUTE,OUTPUT);

delay(1000);
Serial.printf("MIDI notes: %d\n", NOTE_COUNT);
Serial.println("Start!");

//midi.fetchByTrackName("Song.mid", "Recorder");
timebase = millis();
}

void loop() {
  if (millis() - lastServoUpdate >= 10) {
    updateServo(Servos[0]);
    updateServo(Servos[1]);
    updateServo(Servos[2]);
    updateServo(Servos[3]);
    updateServo(Servos[4]);
    updateServo(Servos[5]);
    lastServoUpdate = millis();
  }

  playMidiSong();
}

```

NEW CODE on Tuesday

```
#define MUTE 18
#define HOLE01 1
#define HOLE02 19
#define HOLE1 10
#define HOLE2 4
#define HOLE3 6
#define HOLE4 5
#define HOLE5 7
#define HOLE6 20
#define HOLE7 21

#define freq 50
#define resolution 12
#define PWMCHANNEL0 0
#define PWMCHANNEL1 1
#define PWMCHANNEL2 2
#define PWMCHANNEL3 3
#define PWMCHANNEL4 4
#define PWMCHANNEL5 5

#define closed01 205
#define open01 220
#define closed02 105
#define open02 120
#define closed2 215
#define open2 200
#define closed3 235
#define open3 230
#define closed4 235
#define open4 200
#define closed5 265
#define open5 240

int min_counts = 102;
int max_counts = 501;

struct ServoState {
    int pin;
```

```

    float currentPos;
    float targetPos;
    float speed;
};

struct Pitch {
    int hole01;
    int hole02;
    int hole1;
    int hole2;
    int hole3;
    int hole4;
    int hole5;
    int hole6;
    int hole7;
};

const Pitch NOTE_A5 {closed01,closed02,0,closed2,open3,open4,open5,0,0}; //
MIDI81
const Pitch NOTE_B5 {closed01,closed02,0,open2,open3,open4,open5,0,0};
//MIDI83,24V
const Pitch NOTE_C6
{closed01,closed02,0,closed2,closed3,closed4,closed5,1,1}; // MIDI84,24V
const Pitch NOTE_D6
{closed01,closed02,0,closed2,closed3,closed4,closed5,1,0}; // MIDI86,24V
const Pitch NOTE_E6
{closed01,closed02,0,closed2,closed3,closed4,closed5,0,0}; // MIDI88,24V
const Pitch NOTE_F6
{closed01,closed02,0,closed2,closed3,closed4,open5,0,0}; // MIDI91,24V
const Pitch NOTE_G6 {closed01,closed02,0,closed2,closed3,open4,open5,0,0};
// MIDI93,24V
const Pitch NOTE_A6
{closed01,closed02,0,closed2,open3,closed4,closed5,1,0};
const Pitch NONE {open01,open02,0,open2,open3,open4,open5,0,0};

int Midi_Note[] = {
    64, 72, 72, 72, 64, 64, 72, 72, 72, 69,
    69, 60, 60, 65, 77, 69, 69, 69, 81, 74,
    81, 71, 69, 71, 72, 71, 69, 67, 64, 77,

```

```
    76, 74, 72, 74, 76, 74, 72, 71, 71, 69,  
    71, 72, 71, 69, 67, 64, 77, 76, 74, 72,  
    76, 79, 72, 72, 71, 69, 65, 72, 72, 71,  
    69, 65  
};
```

```
int start_time[] = {  
    0, 357, 5357, 6786, 7143,  
    7500, 8214, 9643, 10357, 11071,  
    12500, 12857, 13214, 13571, 13929,  
    14643, 14821, 15000, 15357, 15714,  
    16071, 27857, 30357, 30536, 30714,  
    30893, 31071, 31250, 31429, 33393,  
    34643, 34821, 35000, 35893, 36071,  
    36250, 37500, 38571, 39286, 41786,  
    41964, 42143, 42321, 42500, 42679,  
    42857, 44821, 46071, 46250, 46429,  
    47500, 47857, 56429, 60714, 61429,  
    62143, 65000, 67857, 72143, 72857,  
    73571, 76429  
};
```

```
int end_time[] = {  
    3392, 3410, 5696, 7124, 7481,  
    7678, 8553, 9981, 10535, 11410,  
    12839, 13196, 13392, 13910, 14267,  
    14731, 14910, 15339, 15696, 16053,  
    16249, 30231, 30526, 30705, 30883,  
    31062, 31240, 31419, 32106, 34580,  
    34812, 34990, 35763, 36062, 36240,  
    37437, 38517, 39249, 41660, 41955,  
    42133, 42312, 42490, 42669, 42847,  
    43535, 46008, 46240, 46419, 47446,  
    47839, 49553, 60499, 61392, 62106,  
    64856, 67714, 71928, 72821, 73535,  
    76285, 79142  
};
```

```
const int NOTE_COUNT = 62;
```

```

uint32_t timebase;
int currentNote = 0;
bool noteActive = false;

ServoState Servos[6];
unsigned long lastServoUpdate = 0;

void moveServo(int servoNumber, int target, int movementTime) {
    int targetClamped = constrain(target, min_counts, max_counts);
    Servos[servoNumber].targetPos = targetClamped;

    float distance = abs(targetClamped - Servos[servoNumber].currentPos);
    int updateTimes = movementTime / 10;
    Servos[servoNumber].speed = distance / updateTimes;

    if (Servos[servoNumber].speed < 0.2)
        Servos[servoNumber].speed = 0.2;
}

void updateServo(ServoState &servo) {
    if (servo.currentPos == servo.targetPos) return;

    if (servo.currentPos < servo.targetPos) {
        servo.currentPos += servo.speed;
        if (servo.currentPos > servo.targetPos) servo.currentPos =
servo.targetPos;
    } else {
        servo.currentPos -= servo.speed;
        if (servo.currentPos < servo.targetPos) servo.currentPos =
servo.targetPos;
    }
    ledcWrite(servo.pin, (int)servo.currentPos);
}

Pitch midiToPitch(int midiNum) {
    switch(midiNum) {
        // case 72: return NOTE_C5;
        // case 74: return NOTE_D5;
        // case 76: return NOTE_E5;
        // case 77: return NOTE_F5;
    }
}

```

```

    // case 79: return NOTE_G5;

    case 81: return NOTE_A5;
    case 83: return NOTE_B5;
    case 84: return NOTE_C6;
    case 86: return NOTE_D6;
    case 88: return NOTE_E6;
    case 89: return NOTE_F6;
    case 91: return NOTE_G6;
    case 93: return NOTE_A6;

    default: return NONE;
}
}

void playMidiSong() {
    unsigned long nowMs = millis();
    int songTimeSec = (nowMs - timebase);

    if(noteActive && songTimeSec >= end_time[currentNote - 1]) {
        Pitch mutePitch = NONE;
        moveServo(0, mutePitch.hole01, 150);
        moveServo(1, mutePitch.hole02, 150);
        moveServo(2, mutePitch.hole2, 150);
        moveServo(3, mutePitch.hole3, 150);
        moveServo(4, mutePitch.hole4, 150);
        moveServo(5, mutePitch.hole5, 150);
        digitalWrite(HOLE1, mutePitch.hole1);
        digitalWrite(HOLE6, mutePitch.hole6);
        digitalWrite(HOLE7, mutePitch.hole7);
        noteActive = false;
    }

    if(currentNote < NOTE_COUNT && songTimeSec >= start_time[currentNote]) {
        Pitch curPitch = midiToPitch(Midi_Note[currentNote]);

        moveServo(0, curPitch.hole01, 150);
        moveServo(1, curPitch.hole02, 150);
        moveServo(2, curPitch.hole2, 150);
        moveServo(3, curPitch.hole3, 150);
    }
}

```

```

        moveServo(4, curPitch.hole4, 150);
        moveServo(5, curPitch.hole5, 150);

        digitalWrite(HOLE1, curPitch.hole1);
        digitalWrite(HOLE6, curPitch.hole6);
        digitalWrite(HOLE7, curPitch.hole7);

        noteActive = true;
        currentNote++;
    }

    if(currentNote >= NOTE_COUNT && !noteActive) {
        currentNote = 0;
        timebase = millis();
    }
}

void setup() {
    delay(10000);
    Serial.begin(921600);

    ledcAttachChannel(HOLE01, freq, resolution, PWMCHANNEL0);
    ledcAttachChannel(HOLE02, freq, resolution, PWMCHANNEL1);
    ledcAttachChannel(HOLE2, freq, resolution, PWMCHANNEL2);
    ledcAttachChannel(HOLE3, freq, resolution, PWMCHANNEL3);
    ledcAttachChannel(HOLE4, freq, resolution, PWMCHANNEL4);
    ledcAttachChannel(HOLE5, freq, resolution, PWMCHANNEL5);

    ledcWrite(HOLE01, open01);
    ledcWrite(HOLE02, open02);
    ledcWrite(HOLE2, open2);
    ledcWrite(HOLE3, open3);
    ledcWrite(HOLE4, open4);
    ledcWrite(HOLE5, open5);
    digitalWrite(HOLE1, 0);
    digitalWrite(HOLE6, 0);
    digitalWrite(HOLE7, 0);
    digitalWrite(MUTE, 0);

    Servos[0] = {HOLE01, open01, open01, 1.0};

```

```

    Servos[1] = {HOLE02, open02, open02, 1.0};
    Servos[2] = {HOLE2, open2, open2, 1.0};
    Servos[3] = {HOLE3, open3, open3, 1.0};
    Servos[4] = {HOLE4, open4, open4, 1.0};
    Servos[5] = {HOLE5, open5, open5, 1.0};

    pinMode(HOLE1, OUTPUT);
    pinMode(HOLE6, OUTPUT);
    pinMode(HOLE7, OUTPUT);
    pinMode(MUTE, OUTPUT);

    delay(1000);
    Serial.printf("MIDI notes: %d\n", NOTE_COUNT);
    Serial.println("Start!");

    timebase = millis();
}

void loop() {
    if (millis() - lastServoUpdate >= 10) {
        updateServo(Servos[0]);
        updateServo(Servos[1]);
        updateServo(Servos[2]);
        updateServo(Servos[3]);
        updateServo(Servos[4]);
        updateServo(Servos[5]);
        lastServoUpdate = millis();
    }

    playMidiSong();
}

```

Entirely with MIDI file (thunder)

Notes can play: 72,74,76,77,79,81

```

#define MUTE 18
#define HOLE01 1
#define HOLE02 19
#define HOLE1 10
#define HOLE2 4

```

```
#define HOLE3 6
#define HOLE4 5
#define HOLE5 7
#define HOLE6 20
#define HOLE7 21

#define freq 50
#define resolution 12
#define PWMCHANNEL0 0
#define PWMCHANNEL1 1
#define PWMCHANNEL2 2
#define PWMCHANNEL3 3
#define PWMCHANNEL4 4
#define PWMCHANNEL5 5

#define closed01 205
#define open01 220
#define closed02 105
#define open02 120
#define closed2 215
#define open2 200
#define closed3 235
#define open3 230
#define closed4 235
#define open4 200
#define closed5 345
#define open5 325

int min_counts = 102;
int max_counts = 501;

struct ServoState {
    int pin;
    float currentPos;
    float targetPos;
    float speed;
};

struct Pitch {
    int hole01;
    int hole02;
    int hole1;
```

```

int hole2;
int hole3;
int hole4;
int hole5;
int hole6;
int hole7;
};

const Pitch NOTE_C5 {closed01,closed02,0,closed2,closed3,closed4,closed5,1,1};
const Pitch NOTE_D5 {closed01,closed02,0,closed2,closed3,closed4,closed5,1,0};
const Pitch NOTE_E5 {closed01,closed02,0,closed2,closed3,closed4,closed5,0,0};
const Pitch NOTE_F5 {closed01,closed02,0,closed2,closed3,closed4,open5,0,0};
const Pitch NOTE_G5 {closed01,closed02,0,closed2,closed3,open4,open5,0,0};
const Pitch NOTE_A5 {closed01,closed02,0,closed2,open3,open4,open5,0,0};
const Pitch NOTE_B5 {closed01,closed02,0,open2,open3,open4,open5,0,0};
const Pitch NOTE_C6 {closed01,closed02,1,closed2,open3,open4,open5,0,0};
const Pitch NOTE_D6 {open01,open02,1,closed2,open3,open4,open5,0,0};
const Pitch NOTE_E6 {closed01,open02,0,closed2,closed3,closed4,closed5,0,0};
const Pitch NOTE_F6 {closed01,open02,1,closed2,closed3,closed4,open5,0,0};
const Pitch NOTE_G6 {closed01,open02,1,closed2,closed3,open4,open5,0,0};
const Pitch NONE {open01,open02,0,open2,open3,open4,open5,0,0};

struct Note {
    Pitch notePitch;
    int startTime;
    int duration;
};

int midiNote[] = {
    64, 72, 72, 72, 64, 64, 72, 72, 72, 69, 69, 60, 60, 65, 77, 69, 69, 69, 81, 74,
    81, 71, 69, 71, 72, 71, 69, 67, 64, 77, 76, 74, 72, 74, 76, 74, 72, 71, 71, 69,
    71, 72, 71, 69, 67, 64, 77, 76, 74, 72, 76, 79, 72, 72, 71, 69, 65, 72, 72, 71,
    69, 65
};

int startTime[] = {
    70714, 71071, 76071, 77500, 77857, 78214, 78929, 80357, 81071, 81786, 83214, 83571,
    83929, 84286, 84643, 85357, 85536, 85714, 86071, 86429,
    86786, 98571, 101071, 101250, 101429, 101607, 101786, 101964, 102143, 104107, 105357,
    105536, 105714, 106607, 106786, 106964, 108214, 109286, 110000, 112500,
    112679, 112857, 113036, 113214, 113393, 113571, 115536, 116786, 116964, 117143,
    118214, 118571, 127143, 131429, 132143, 132857, 135714, 138571, 142857, 143571,

```

```

144286, 147143
};

int noteDuration[] = {3571, 3214, 356, 356, 356, 356, 356, 356, 356, 356, 356, 356, 356,
356, 356, 356, 178, 178, 356, 356, 356, 356, 2499, 178, 178, 178,
178, 178, 178, 714, 1249, 178, 178, 803, 178, 178, 1249, 1071, 714, 2499, 178, 178,
178, 178, 178, 178, 714, 1249, 178, 178, 1071, 356, 1785, 4285,
714, 714, 2856, 2856, 4285, 714, 714, 2856, 2856
};

int sizeofMidi = sizeof(midiNote) / sizeof(midiNote[0]);
Note song[sizeof(midiNote) / sizeof(midiNote[0])];
int currentNote = 0;
ServoState Servos[6];
bool playing = true;

uint32_t timebase;
unsigned long lastServoUpdate = 0;

void setup() {
    delay(5000);
    Serial.begin(921600);
    Serial.println("Entering setup");

    for (int i = 0; i < sizeofMidi; i++) {
        song[i] = {getPitch(getNoteName(midiNote[i])), startTime[i]-60000, noteDuration[i]};
    }

    ledcAttachChannel(HOLE01, freq, resolution, PWMCHANNEL0);
    ledcAttachChannel(HOLE02, freq, resolution, PWMCHANNEL1);
    ledcAttachChannel(HOLE2, freq, resolution, PWMCHANNEL2);
    ledcAttachChannel(HOLE3, freq, resolution, PWMCHANNEL3);
    ledcAttachChannel(HOLE4, freq, resolution, PWMCHANNEL4);
    ledcAttachChannel(HOLE5, freq, resolution, PWMCHANNEL5);

    ledcWrite(HOLE01, 220);
    ledcWrite(HOLE02, 120);
    ledcWrite(HOLE2, 200);
    ledcWrite(HOLE3, 230);
    ledcWrite(HOLE4, 200);
    ledcWrite(HOLE5, 325);

```

```

Servos[0] = {HOLE01, 220, 220, 1.0};
Servos[1] = {HOLE02, 120, 120, 1.0};
Servos[2] = {HOLE2, 200, 200, 1.0};
Servos[3] = {HOLE3, 230, 230, 1.0};
Servos[4] = {HOLE4, 200, 200, 1.0};
Servos[5] = {HOLE5, 325, 325, 1.0};

pinMode(HOLE1, OUTPUT);
pinMode(HOLE6, OUTPUT);
pinMode(HOLE7, OUTPUT);
pinMode(MUTE, OUTPUT);

digitalWrite(HOLE1, LOW);
digitalWrite(HOLE6, LOW);
digitalWrite(HOLE7, LOW);
digitalWrite(MUTE, LOW);

delay(1000);
Serial.println("code started");
timebase = millis();
}

void loop() {
  if (millis() - lastServoUpdate >= 10) {
    updateServo(Servos[0]);
    updateServo(Servos[1]);
    updateServo(Servos[2]);
    updateServo(Servos[3]);
    updateServo(Servos[4]);
    updateServo(Servos[5]);
    lastServoUpdate = millis();
  }
  playSong();
}

void playSong() {
  if (!playing) return;

  if (millis() - timebase > song[currentNote].startTime) {
    Pitch curPitch = song[currentNote].notePitch;

    if (!isNone(curPitch)) {

```

```

        moveServo(0, curPitch.hole01, 150);
        moveServo(1, curPitch.hole02, 150);
        moveServo(2, curPitch.hole2, 150);
        moveServo(3, curPitch.hole3, 150);
        moveServo(4, curPitch.hole4, 150);
        moveServo(5, curPitch.hole5, 150);

        digitalWrite(HOLE1, curPitch.hole1);
        digitalWrite(HOLE6, curPitch.hole6);
        digitalWrite(HOLE7, curPitch.hole7);
    }

    if (millis() - song[currentNote].startTime > song[currentNote].duration ||
isNone(curPitch)) {
        digitalWrite(MUTE, HIGH);
    } else {
        digitalWrite(MUTE, LOW);
    }

    if (millis() - song[currentNote].startTime > song[currentNote].duration) {
        currentNote++;
    }

    if (currentNote >= sizeofMidi) {
        playing = false;
        currentNote = 0;
        timebase = millis();
    }
}
}

void moveServo(int servoNumber, int target, int movementTime) {
    int targetClamped = constrain(target, min_counts, max_counts);
    Servos[servoNumber].targetPos = targetClamped;

    float distance = abs(targetClamped - Servos[servoNumber].currentPos);
    int updateTimes = movementTime / 10;
    Servos[servoNumber].speed = distance / updateTimes;

    if (Servos[servoNumber].speed < 0.2)
        Servos[servoNumber].speed = 0.2;
}

```

```

void updateServo(ServoState &servo) {
    if (servo.currentPos == servo.targetPos) return;

    if (servo.currentPos < servo.targetPos) {
        servo.currentPos += servo.speed;
        if (servo.currentPos > servo.targetPos) servo.currentPos = servo.targetPos;
    } else {
        servo.currentPos -= servo.speed;
        if (servo.currentPos < servo.targetPos) servo.currentPos = servo.targetPos;
    }
    ledcWrite(servo.pin, (int)servo.currentPos);
}

```

```

Pitch getNoteName(int midi) {
    switch (midi) {
        case 72: return NOTE_C5;
        // case 73: return NOTE_CS5;
        case 74: return NOTE_D5;
        // case 75: return NOTE_DS5;
        case 76: return NOTE_E5;
        case 77: return NOTE_F5;
        // case 78: return NOTE_FS5;
        case 79: return NOTE_G5;
        // case 80: return NOTE_GS5;
        case 81: return NOTE_A5;
        // case 82: return NOTE_AS5;
        // case 83: return NOTE_B5;
        // case 84: return NOTE_C6;
        // case 85: return NOTE_CS6;
        // case 86: return NOTE_D6;
        // case 87: return NOTE_DS6;
        // case 88: return NOTE_E6;
        // case 89: return NOTE_F6;
        // case 90: return NOTE_FS6;
        // case 91: return NOTE_G6;
        // case 92: return NOTE_GS6;
        // case 93: return NOTE_A6;
        // case 94: return NOTE_AS6;
        // case 95: return NOTE_B6;
        // case 96: return NOTE_C7;
        default: return NONE;
    }
}

```

```

}
}

bool isNone(Pitch p) {
    return (
        p.hole01 == NONE.hole01 &&
        p.hole02 == NONE.hole02 &&
        p.hole1 == NONE.hole1 &&
        p.hole2 == NONE.hole2 &&
        p.hole3 == NONE.hole3 &&
        p.hole4 == NONE.hole4 &&
        p.hole5 == NONE.hole5 &&
        p.hole6 == NONE.hole6 &&
        p.hole7 == NONE.hole7
    );
}

```

According to MIDI file (start time & duration), along with MUTE

```

#define MUTE 18
#define HOLE01 1
#define HOLE02 0
#define HOLE1 10
#define HOLE2 4
#define HOLE3 6
#define HOLE4 5
#define HOLE5 7
#define HOLE6 20
#define HOLE7 21

#define freq 50
#define resolution 12
#define PWMCHANNEL0 0
#define PWMCHANNEL1 1
#define PWMCHANNEL2 2
#define PWMCHANNEL3 3
#define PWMCHANNEL4 4
#define PWMCHANNEL5 5

```

```
#define closed01 205
#define open01 220
#define closed02 105
#define open02 120
#define closed2 215
#define open2 200
#define closed3 235
#define open3 230
#define closed4 235
#define open4 200
#define closed5 345
#define open5 325

int min_counts = 102;
int max_counts = 501;

struct ServoState {
    int pin;
    float currentPos;
    float targetPos;
    float speed;
};

struct Pitch {
    int hole01;
    int hole02;
    int hole1;
    int hole2;
    int hole3;
    int hole4;
    int hole5;
    int hole6;
    int hole7;
};

// const Pitch NOTE_F4 = {closed01, closed02, 0, closed2, closed3, closed4, closed5,
1, 1};
// const Pitch NOTE_G4 = {closed01, closed02, 0, closed2, closed3, closed4, closed5,
1, 0};
// const Pitch NOTE_A4 = {closed01, closed02, 0, closed2, closed3, closed4, closed5,
0, 0};
```

```

// const Pitch NOTE_B4 = {closed01, closed02, 0, closed2, closed3, open4, closed5, 1,
0};
// const Pitch NOTE_C5 = {closed01, closed02, 0, closed2, closed3, open4, open5, 0,
0};
// const Pitch NOTE_D5 = {closed01, closed02, 0, closed2, open3, open4, open5, 0, 0};
// const Pitch NOTE_E5 = {closed01, closed02, 0, open2, open3, open4, open5, 0, 0};
// const Pitch NOTE_F5 = {closed01, closed02, 1, closed2, open3, open4, open5, 0, 0};
// const Pitch NOTE_G5 = {open01, open02, 0, closed2, closed3, closed4, closed5, 1,
1};
// const Pitch NOTE_A5 = {closed01, open02, 0, closed2, closed3, closed4, closed5, 0,
0};

// const Pitch NOTE_A5{closed01,closed02,0,closed2,open3,open4,open5,0,0};
// const Pitch NOTE_As5{closed01,closed02,0,open2,closed3,open4,closed5,1,1};
// const Pitch NOTE_B5{closed01,closed02,0,open2,open3,open4,open5,0,0};
// const Pitch NOTE_C6{closed01,closed02,1,closed2,open3,open4,open5,0,0};
// const Pitch NOTE-Cs6{open01,open02,0,open2,open3,open4,open5,0,0};
// const Pitch NOTE_D6{open01,open02,1,open2,open3,open4,open5,0,0};
// E6{closed01,closed02,0,closed2,closed3,closed4,closed5,0,0}
// const Pitch NOTE_F6 = {closed01,closed02,0,closed2,closed3,closed4,open5,0,0};
// G6{closed01,closed02,0,closed2,closed3,open4,open5,0,0}

const Pitch NOTE_C5{closed01,closed02,0,closed2,closed3,closed4,closed5,1,1};
const Pitch NOTE_D5{closed01,closed02,0,closed2,closed3,closed4,closed5,1,0};
const Pitch NOTE_E5{closed01,closed02,0,closed2,closed3,closed4,closed5,0,0};
const Pitch NOTE_F5{closed01,closed02,0,closed2,closed3,closed4,open5,0,0};
const Pitch NOTE_G5{closed01,closed02,0,closed2,closed3,open4,open5,0,0};
const Pitch NOTE_A5{closed01,closed02,0,closed2,open3,open4,open5,0,0};
const Pitch NOTE_B5{closed01,closed02,0,open2,open3,open4,open5,0,0};
const Pitch NOTE_C6{closed01,closed02,1,closed2,open3,open4,open5,0,0};
const Pitch NOTE_D6{open01,open02,1,closed2,open3,open4,open5,0,0};
const Pitch NOTE_E6{closed01,open02,0,closed2,closed3,closed4,closed5,0,0};
const Pitch NOTE_F6{closed01,open02,1,closed2,closed3,closed4,open5,0,0};
const Pitch NOTE_G6{closed01,open02,1,closed2,closed3,open4,open5,0,0};

struct Note {
    Pitch notePitch;
    int startTime;
    int duration;
};

```

```
Note song[] = {
    {NOTE_C5, 1000, 500},
    {NOTE_D5, 2000, 500},
    {NOTE_E5, 3000, 500},
    {NOTE_F5, 4000, 500},
    {NOTE_G5, 5000, 500},
    {NOTE_A5, 6000, 500},
    {NOTE_B5, 7000, 500}
};

const int songLength = sizeof(song) / sizeof(song[0]);
int currentNote = 0;

ServoState Servos[6];

bool playing = true;

uint32_t timebase;
unsigned long lastServoUpdate = 0;
unsigned long noteStartTime = 0;

void setup() {
    delay(5000);
    Serial.begin(921600);
    Serial.println("Entering setup");

    ledcAttachChannel(HOLE01, freq, resolution, PWMCHANNEL0);
    ledcAttachChannel(HOLE02, freq, resolution, PWMCHANNEL1);
    ledcAttachChannel(HOLE2, freq, resolution, PWMCHANNEL2);
    ledcAttachChannel(HOLE3, freq, resolution, PWMCHANNEL3);
    ledcAttachChannel(HOLE4, freq, resolution, PWMCHANNEL4);
    ledcAttachChannel(HOLE5, freq, resolution, PWMCHANNEL5);

    ledcWrite(HOLE01, 220);
    ledcWrite(HOLE02, 120);
    ledcWrite(HOLE2, 200);
    ledcWrite(HOLE3, 230);
    ledcWrite(HOLE4, 200);
    ledcWrite(HOLE5, 325);

    Servos[0] = {HOLE01, 220, 220, 1.0};
    Servos[1] = {HOLE02, 120, 120, 1.0};
```

```

Servos[2] = {HOLE2, 200, 200, 1.0};
Servos[3] = {HOLE3, 230, 230, 1.0};
Servos[4] = {HOLE4, 200, 200, 1.0};
Servos[5] = {HOLE5, 325, 325, 1.0};

pinMode(HOLE1, OUTPUT);
pinMode(HOLE6, OUTPUT);
pinMode(HOLE7, OUTPUT);
pinMode(MUTE, OUTPUT);

digitalWrite(HOLE1, LOW);
digitalWrite(HOLE6, LOW);
digitalWrite(HOLE7, LOW);
digitalWrite(MUTE, OUTPUT);

delay(1000);
Serial.println("code started");
timebase = millis();
}

void loop() {
  if (millis() - lastServoUpdate >= 10) {
    updateServo(Servos[0]);
    updateServo(Servos[1]);
    updateServo(Servos[2]);
    updateServo(Servos[3]);
    updateServo(Servos[4]);
    updateServo(Servos[5]);
    lastServoUpdate = millis();
  }
  playSong();
}

void playSong() {
  if (!playing) return;

  if (millis() - timebase > song[currentNote].startTime) {
    noteStartTime = millis();
    Pitch curPitch = song[currentNote].notePitch;

    moveServo(0, curPitch.hole01, 150);
    moveServo(1, curPitch.hole02, 150);
  }
}

```

```

    moveServo(2, curPitch.hole2, 150);
    moveServo(3, curPitch.hole3, 150);
    moveServo(4, curPitch.hole4, 150);
    moveServo(5, curPitch.hole5, 150);

    digitalWrite(HOLE1, curPitch.hole1);
    digitalWrite(HOLE6, curPitch.hole6);
    digitalWrite(HOLE7, curPitch.hole7);

    if (millis() - noteStartTime > song[currentNote].duration) {
        digitalWrite(MUTE,1);
    } else {
        digitalWrite(MUTE,0);
    }

    currentNote++;
    if (currentNote >= songLength) {
        playing = false;
        currentNote = 0;
        timebase = millis();
    }
}
}

void moveServo(int servoNumber, int target, int movementTime) {
    int targetClamped = constrain(target, min_counts, max_counts);
    Servos[servoNumber].targetPos = targetClamped;

    float distance = abs(targetClamped - Servos[servoNumber].currentPos);
    int updateTimes = movementTime / 10;
    Servos[servoNumber].speed = distance / updateTimes;

    if (Servos[servoNumber].speed < 0.2)
        Servos[servoNumber].speed = 0.2;
}

void updateServo(ServoState &servo) {
    if (servo.currentPos == servo.targetPos) return;

    if (servo.currentPos < servo.targetPos) {
        servo.currentPos += servo.speed;
        if (servo.currentPos > servo.targetPos) servo.currentPos = servo.targetPos;
    }
}

```

```

} else {
    servo.currentPos -= servo.speed;
    if (servo.currentPos < servo.targetPos) servo.currentPos = servo.targetPos;
}
ledcWrite(servo.pin, (int)servo.currentPos);
}

```

First (working) iterations via targetPosition();

```

#define HOLE1 5
#define HOLE2 4
#define freq 50
#define resolution 12
#define PWMCHANNEL1 1
#define PWMCHANNEL2 2
#define startingPosition1 150
#define startingPosition2 150
#define ONEms_COUNTS ((1 << resolution) * 50 / 1000)

int min_counts = 102;
int max_counts = 501;
int currentPosition[] = {0, 0, 0, 0, startingPosition1, startingPosition2};

void setup() {
    ledcAttachChannel(HOLE1, freq, resolution, PWMCHANNEL1);
    ledcAttachChannel(HOLE2, freq, resolution, PWMCHANNEL2);
    Serial.begin(115200);
    ledcWrite(HOLE1, startingPosition1);
    ledcWrite(HOLE2, startingPosition2);
    delay(1000);
}

void loop() {
    targetPosition(210, 210, 150);
    delay(100);
    targetPosition(180, 190, 150);
    delay(100);
}

void targetPosition(int targetAngle1, int targetAngle2, int movementTime) {

```

```

float steps = 20.0;
int current1 = currentPosition[HOLE1];
int current2 = currentPosition[HOLE2];

float change1 = (targetAngle1 - current1) / steps;
float change2 = (targetAngle2 - current2) / steps;

for (int i = 0; i < steps; i++) {
    current1 += change1;
    current2 += change2;
    ledcWrite(HOLE1, current1);
    ledcWrite(HOLE2, current2);
    delayMicroseconds(movementTime * 1000 / steps);
}

currentPosition[HOLE1] = targetAngle1;
currentPosition[HOLE2] = targetAngle2;
}

```

Doesn't work in setup but works otherwise:

```

#define HOLE1 4
#define HOLE2 5
#define freq 50
#define resolution 12
#define PWMCHANNEL1 1
#define PWMCHANNEL2 2
#define startingPosition1 150
#define startingPosition2 100

#define closed1 210
#define open1 180
#define closed2 180
#define open2 160

int min_counts = 102;
int max_counts = 501;

struct ServoState {
    int pin;
    float currentPos;
    float targetPos;
}

```

```

    float speed;
};

struct Note {
    int hole1;
    int hole2;
    int duration;
};

Note song[] = {
    {closed1, open2, 1000},
    {open1, closed2, 1000},
    {closed1, open2, 1000},
    {open1, closed2, 1000},
    {open1, open2, 1000},
    {closed1, closed2, 1000},
    {closed1, open2, 1000},
    {open1, closed2, 1000},
    {closed1, open2, 1000},
    {open1, closed2, 1000}
};

const int songLength = sizeof(song) / sizeof(song[0]);
int currentNote = 0;

ServoState Servos[2];
bool playing = true;
unsigned long noteStartTime = 0;
unsigned long lastServoUpdate = 0;

void setup() {
    Serial.begin(115200);
    Serial.println("Entering setup");

    ledcAttachChannel(HOLE1, freq, resolution, PWMCHANNEL1);
    ledcAttachChannel(HOLE2, freq, resolution, PWMCHANNEL2);

    Servos[0] = {HOLE1, 200, 200, 1.0};
    Servos[1] = {HOLE2, 200, 200, 1.0};

    ledcWrite(HOLE1, 150);
    ledcWrite(HOLE2, 150);

```

```

    delay(1000);
    Serial.println("code started");
}

void loop() {
    while(1)
    {
        Serial.println("print");
        delay(1000);
    }

    updateServo(Servos[0]);
    updateServo(Servos[1]);
    playSong();
}

void playSong() {
    if (!playing) {
        return;
    }

    if (millis() - noteStartTime > song[currentNote].duration) {
        moveServo(0, song[currentNote].hole1, 150);
        moveServo(1, song[currentNote].hole2, 150);

        noteStartTime = millis();

        if (currentNote >= songLength) {
            playing = false;
            return;
        }

        currentNote++;
    }
}

void moveServo(int servoNumber, int target, int movementTime) {
    Servos[servoNumber].targetPos = target;

    float distance = abs(target - Servos[servoNumber].currentPos);
    float updates = movementTime / 10.0;

```

```

Servos[servoNumber].speed = distance / updates;

if (Servos[servoNumber].speed < 0.2) {
    Servos[servoNumber].speed = 0.2;
}
}

void updateServo(ServoState &servo) {
    if (servo.currentPos < servo.targetPos) {
        servo.currentPos += servo.speed;

        if (servo.currentPos > servo.targetPos)
            servo.currentPos = servo.targetPos;
    }
    else if (servo.currentPos > servo.targetPos) {
        servo.currentPos -= servo.speed;

        if (servo.currentPos < servo.targetPos) {
            servo.currentPos = servo.targetPos;
        }
    }
}

ledcWrite(servo.pin, (int)servo.currentPos);
}

```

First fully function iteration w/ missing values

```

#define HOLE01
#define HOLE02
#define HOLE1
#define HOLE2
#define HOLE3
#define HOLE4
#define HOLE5
#define HOLE6
#define HOLE7
#define freq 50
#define resolution 12
#define PWMCHANNEL0 0
#define PWMCHANNEL1 1
#define PWMCHANNEL2 2

```

```
#define PWMCHANNEL3 3
#define PWMCHANNEL4 4
#define PWMCHANNEL5 5
#define startingPosition1 150
#define startingPosition2 100
```

```
#define closed01
#define open01
#define closed02
#define open02
#define closed2
#define open2
#define closed3
#define open3
#define closed4
#define open4
#define closed5
#define open6
```

```
int min_counts = 102;
int max_counts = 501;
```

```
struct ServoState {
    int pin;
    float currentPos;
    float targetPos;
    float speed;
};
```

```
struct Pitch {
    int hole01;
    int hole02;
    int hole1;
    int hole2;
    int hole3;
    int hole4;
    int hole5;
    int hole6;
    int hole7;
}
```

```

Pitch F4 = {closed01,closed02,0,closed2,closed3,closed4,open5,1,1};
Pitch G4 = {closed01,closed02,0,closed2,closed3,open4,open5,0,0};
Pitch A4 = {closed01,closed02,0,closed2,open3,open4,open5,0,0};
Pitch B4 = {closed01,closed02,0,open2,open3,open4,open5,0,0};
Pitch C5 = {closed01,closed02,1,closed2,open3,open4,open5,0,0};
Pitch D5 = {open01,open02,0,closed2,closed3,closed4,closed5,1,1};
Pitch E5 = {closed01,open02,0,closed2,closed3,closed4,closed5,0,0};
Pitch F5 = {closed01,open02,0,closed2,closed3,closed4,open5,1,0};
Pitch G5 = {closed01,open02,0,closed2,closed3,open4,open5,0,0};
Pitch A5 = {closed01,open02,0,closed2,open3,open4,open5,0,0};

```

```

struct Note {
    Pitch notePitch;
    int duration;
};

```

```

Note song[] = {
    {F4, 1000},
    {G4, 1000},
    {A4, 1000},
    {B4, 1000},
    {C5, 1000},
    {D5, 1000},
    {E5, 1000},
    {F5, 1000},
    {G5, 1000},
    {A5, 1000}
};

```

```

const int songLength = sizeof(song) / sizeof(song[0]);
int currentNote = 0;
ServoState Servos[6];
bool playing = true;

```

```

unsigned long noteStartTime = 0;
unsigned long lastServoUpdate = 0;

```

```

void setup() {
    Serial.begin(115200);
    Serial.println("Entering setup");

    ledcAttachChannel(HOLE01,freq,resolution,PWMCHANNEL0);
}

```

```

ledcAttachChannel(HOLE02,freq,resolution,PWMCHANNEL1);
ledcAttachChannel(HOLE2,freq,resolution,PWMCHANNEL2);
ledcAttachChannel(HOLE3,freq,resolution,PWMCHANNEL3);
ledcAttachChannel(HOLE4,freq,resolution,PWMCHANNEL4);
ledcAttachChannel(HOLE5,freq,resolution,PWMCHANNEL5);

ledcWrite(HOLE01, 130);
ledcWrite(HOLE02, 130);
ledcWrite(HOLE2, 130);
ledcWrite(HOLE3, 130);
ledcWrite(HOLE4, 130);
ledcWrite(HOLE5, 130);

Servos[0] = {HOLE01, 130, 130, 1.0};
Servos[1] = {HOLE02, 130, 130, 1.0};
Servos[2] = {HOLE2, 130, 130, 1.0};
Servos[3] = {HOLE3, 130, 130, 1.0};
Servos[4] = {HOLE4, 130, 130, 1.0};
Servos[5] = {HOLE5, 130, 130, 1.0};

pinMode(HOLE1,OUTPUT);
pinMode(HOLE6,OUTPUT);
pinMode(HOLE7,OUTPUT);

delay(1000);
Serial.println("code started");
}

void loop() {
  if (millis() - lastServoUpdate >= 10) {
    updateServo(Servos[0]);
    updateServo(Servos[1]);
    updateServo(Servos[2]);
    updateServo(Servos[3]);
    updateServo(Servos[4]);
    updateServo(Servos[5]);
    lastServoUpdate = millis();
  }
  playSong();
}

void playSong() {

```

```

if (!playing) {
    return;
}

if (millis() - noteStartTime > song[currentNote].duration) {
    noteStartTime = millis();
    moveServo(0, song[currentNote].hole01, 150);
    moveServo(1, song[currentNote].hole02, 150);
    moveServo(2, song[currentNote].hole2, 150);
    moveServo(3, song[currentNote].hole3, 150);
    moveServo(4, song[currentNote].hole4, 150);
    moveServo(5, song[currentNote].hole5, 150);

    digitalWrite(HOLE1, song[currentNote].notePitch.hole1);
    digitalWrite(HOLE6, song[currentNote].notePitch.hole6);
    digitalWrite(HOLE7, song[currentNote].notePitch.hole7);

    currentNote++;
    if (currentNote >= songLength) {
        playing = false;
        currentNote = 0;
        return;
    }
}

void moveServo(int servoNumber, int target, int movementTime) {
    Servos[servoNumber].targetPos = target;

    float distance = abs(target - Servos[servoNumber].currentPos);
    int updates = movementTime / 10;

    Servos[servoNumber].speed = distance / updates;

    if (Servos[servoNumber].speed < 0.2) {
        Servos[servoNumber].speed = 0.2;
    }
}

void updateServo(ServoState &servo) {
    if (servo.currentPos == servo.targetPos) {
        return;
    }
}

```

```

}
else if (servo.currentPos < servo.targetPos) {
    servo.currentPos += servo.speed;

    if (servo.currentPos > servo.targetPos)
        servo.currentPos = servo.targetPos;
}
else {
    servo.currentPos -= servo.speed;

    if (servo.currentPos < servo.targetPos) {
        servo.currentPos = servo.targetPos;
    }
}
ledcWrite(servo.pin, (int)servo.currentPos);
}

```

Code with 7, 72 servos added:

```

#define HOLE01
#define HOLE02
#define HOLE1
#define HOLE2
#define HOLE3
#define HOLE4
#define HOLE5
#define HOLE6
#define HOLE7
#define HOLE72
#define freq 50
#define resolution 12
#define PWMCHANNEL0 0
#define PWMCHANNEL1 1
#define PWMCHANNEL2 2
#define PWMCHANNEL3 3
#define PWMCHANNEL4 4
#define PWMCHANNEL5 5

#define closed01
#define open01
#define closed02
#define open02

```

```

#define closed2 200
#define open2 240
#define closed3 220
#define open3 260
#define closed4 180
#define open4 224
#define closed5 330
#define open5 380

int min_counts = 102;
int max_counts = 501;

struct ServoState {
    int pin;
    float currentPos;
    float targetPos;
    float speed;
};

struct Pitch {
    int hole01;
    int hole02;
    int hole1;
    int hole2;
    int hole3;
    int hole4;
    int hole5;
    int hole6;
    int hole7;
}

Pitch F4 = {closed01,closed02,0,closed2,closed3,closed4,open5,1,1};
Pitch G4 = {closed01,closed02,0,closed2,closed3,open4,open5,0,0};
Pitch A4 = {closed01,closed02,0,closed2,open3,open4,open5,0,0};
Pitch B4 = {closed01,closed02,0,open2,open3,open4,open5,0,0};
Pitch C5 = {closed01,closed02,1,closed2,open3,open4,open5,0,0};
Pitch D5 = {open01,open02,0,closed2,closed3,closed4,closed5,1,1};
Pitch E5 = {closed01,open02,0,closed2,closed3,closed4,closed5,0,0};
Pitch F5 = {closed01,open02,0,closed2,closed3,closed4,open5,1,0};
Pitch G5 = {closed01,open02,0,closed2,closed3,open4,open5,0,0};
Pitch A5 = {closed01,open02,0,closed2,open3,open4,open5,0,0};

```

```

struct Note {
    Pitch notePitch;
    int duration;
};

Note song[] = {
    {F4, 1000},
    {G4, 1000},
    {A4, 1000},
    {B4, 1000},
    {C5, 1000},
    {D5, 1000},
    {E5, 1000},
    {F5, 1000},
    {G5, 1000},
    {A5, 1000}
};

const int songLength = sizeof(song) / sizeof(song[0]);
int currentNote = 0;

ServoState Servos[6];

bool playing = true;
unsigned long noteStartTime = 0;
unsigned long lastServoUpdate = 0;

void setup() {
    Serial.begin(115200);
    Serial.println("Entering setup");

    ledcAttachChannel(HOLE01, freq, resolution, PWMCHANNEL0);
    ledcAttachChannel(HOLE02, freq, resolution, PWMCHANNEL1);
    ledcAttachChannel(HOLE2, freq, resolution, PWMCHANNEL2);
    ledcAttachChannel(HOLE3, freq, resolution, PWMCHANNEL3);
    ledcAttachChannel(HOLE4, freq, resolution, PWMCHANNEL4);
    ledcAttachChannel(HOLE5, freq, resolution, PWMCHANNEL5);

    ledcWrite(HOLE01, 150);
    ledcWrite(HOLE02, 150);
    ledcWrite(HOLE2, 150);
    ledcWrite(HOLE3, 150);

```

```

    ledcWrite(HOLE4, 150);
    ledcWrite(HOLE5, 150);

    Servos[0] = {HOLE01, 150, 150, 1.0};
    Servos[1] = {HOLE02, 150, 150, 1.0};
    Servos[2] = {HOLE2, 150, 150, 1.0};
    Servos[3] = {HOLE3, 150, 150, 1.0};
    Servos[4] = {HOLE4, 150, 150, 1.0};
    Servos[5] = {HOLE5, 150, 150, 1.0};

    pinMode(HOLE1, OUTPUT);
    pinMode(HOLE6, OUTPUT);
    pinMode(HOLE7, OUTPUT);
    pinMode(HOLE72, OUTPUT);

    delay(1000);
    Serial.println("code started");
}

void loop() {
    if (millis() - lastServoUpdate >= 10) {
        updateServo(Servos[0]);
        updateServo(Servos[1]);
        updateServo(Servos[2]);
        updateServo(Servos[3]);
        updateServo(Servos[4]);
        updateServo(Servos[5]);
        lastServoUpdate = millis();
    }
    playSong();
}

void playSong() {
    if (!playing) {
        return;
    }

    if (millis() - noteStartTime > song[currentNote].duration) {
        noteStartTime = millis();
        moveServo(0, song[currentNote].hole01, 150);
        moveServo(1, song[currentNote].hole02, 150);
        moveServo(2, song[currentNote].hole2, 150);
    }
}

```

```

    moveServo(3, song[currentNote].hole3, 150);
    moveServo(4, song[currentNote].hole4, 150);
    moveServo(5, song[currentNote].hole5, 150);

    digitalWrite(HOLE1, song[currentNote].notePitch.hole1);
    digitalWrite(HOLE6, song[currentNote].notePitch.hole6);
    digitalWrite(HOLE7, song[currentNote].notePitch.hole7);
    digitalWrite(HOLE72, song[currentNote].notePitch.hole7);

    currentNote++;
    if (currentNote >= songLength) {
        playing = false;
        currentNote = 0;
        return;
    }
}
}

void moveServo(int servoNumber, int target, int movementTime) {
    Servos[servoNumber].targetPos = target;

    float distance = abs(target - Servos[servoNumber].currentPos);
    int updates = movementTime / 10;

    Servos[servoNumber].speed = distance / updates;

    if (Servos[servoNumber].speed < 0.2) {
        Servos[servoNumber].speed = 0.2;
    }
}

void updateServo(ServoState &servo) {
    if (servo.currentPos == servo.targetPos) {
        return;
    }
    else if (servo.currentPos < servo.targetPos) {
        servo.currentPos += servo.speed;

        if (servo.currentPos > servo.targetPos)
            servo.currentPos = servo.targetPos;
    }
    else {

```

```
servo.currentPos -= servo.speed;

if (servo.currentPos < servo.targetPos) {
    servo.currentPos = servo.targetPos;
}
}
ledcWrite(servo.pin, (int)servo.currentPos);
}
```

Code with working servos but problematic solenoids

```
#define HOLE01 1
#define HOLE02 0
#define HOLE1 10
#define HOLE2 4
#define HOLE3 6
#define HOLE4 5
#define HOLE5 7
#define HOLE6 20
#define HOLE7 21

#define freq 50
#define resolution 12
#define PWMCHANNEL0 0
#define PWMCHANNEL1 1
#define PWMCHANNEL2 2
#define PWMCHANNEL3 3
#define PWMCHANNEL4 4
#define PWMCHANNEL5 5

#define closed01 205
#define open01 220
#define closed02 105
#define open02 120
#define closed2 215
#define open2 200
#define closed3 235
#define open3 230
#define closed4 235
#define open4 200
```

```
#define closed5 345
#define open5 325

int min_counts = 102;
int max_counts = 501;

struct ServoState {
    int pin;
    float currentPos;
    float targetPos;
    float speed;
};

struct Pitch {
    int hole01;
    int hole02;
    int hole1;
    int hole2;
    int hole3;
    int hole4;
    int hole5;
    int hole6;
    int hole7;
};

// const Pitch NOTE_F4 = {closed01, closed02, 0, closed2, closed3,
closed4, closed5, 1, 1};
// const Pitch NOTE_G4 = {closed01, closed02, 0, closed2, closed3,
closed4, closed5, 1, 0};
// const Pitch NOTE_A4 = {closed01, closed02, 0, closed2, closed3,
closed4, closed5, 0, 0};
// const Pitch NOTE_B4 = {closed01, closed02, 0, closed2, closed3, open4,
closed5, 1, 0};
// const Pitch NOTE_C5 = {closed01, closed02, 0, closed2, closed3, open4,
open5, 0, 0};
// const Pitch NOTE_D5 = {closed01, closed02, 0, closed2, open3, open4,
open5, 0, 0};
// const Pitch NOTE_E5 = {closed01, closed02, 0, open2, open3, open4,
open5, 0, 0};
```

```

// const Pitch NOTE_F5 = {closed01, closed02, 1, closed2, open3, open4,
open5, 0, 0};
// const Pitch NOTE_G5 = {open01, open02, 0, closed2, closed3, closed4,
closed5, 1, 1};
// const Pitch NOTE_A5 = {closed01, open02, 0, closed2, closed3, closed4,
closed5, 0, 0};

// const Pitch NOTE_A5{closed01,closed02,0,closed2,open3,open4,open5,0,0};
// const Pitch
NOTE_As5{closed01,closed02,0,open2,closed3,open4,closed5,1,1};
// const Pitch NOTE_B5{closed01,closed02,0,open2,open3,open4,open5,0,0};
// const Pitch NOTE_C6{closed01,closed02,1,closed2,open3,open4,open5,0,0};
// const Pitch NOTE-Cs6{open01,open02,0,open2,open3,open4,open5,0,0};
// const Pitch NOTE_D6{open01,open02,1,open2,open3,open4,open5,0,0};
// E6{closed01,closed02,0,closed2,closed3,closed4,closed5,0,0}
// const Pitch NOTE_F6 =
{closed01,closed02,0,closed2,closed3,closed4,open5,0,0};
// G6{closed01,closed02,0,closed2,closed3,open4,open5,0,0}

const Pitch
NOTE_C5{closed01,closed02,0,closed2,closed3,closed4,closed5,1,1};
const Pitch
NOTE_D5{closed01,closed02,0,closed2,closed3,closed4,closed5,1,0};
const Pitch
NOTE_E5{closed01,closed02,0,closed2,closed3,closed4,closed5,0,0};
const Pitch
NOTE_F5{closed01,closed02,0,closed2,closed3,closed4,open5,0,0};
const Pitch NOTE_G5{closed01,closed02,0,closed2,closed3,open4,open5,0,0};
const Pitch NOTE_A5{closed01,closed02,0,closed2,open3,open4,open5,0,0};
const Pitch NOTE_B5{closed01,closed02,0,open2,open3,open4,open5,0,0};
const Pitch NOTE_C6{closed01,closed02,1,closed2,open3,open4,open5,0,0};
const Pitch NOTE_D6{open01,open02,1,closed2,open3,open4,open5,0,0};
const Pitch
NOTE_E6{closed01,open02,0,closed2,closed3,closed4,closed5,0,0};
const Pitch NOTE_F6{closed01,open02,1,closed2,closed3,closed4,open5,0,0};
const Pitch NOTE_G6{closed01,open02,1,closed2,closed3,open4,open5,0,0};

struct Note {
    Pitch notePitch;

```

```

    int duration;
};

Note song[] = {
    {NOTE_C5, 1000},
    {NOTE_D5, 1000},
    {NOTE_E5, 1000},
    {NOTE_F5, 1000},
    {NOTE_G5, 1000},
    {NOTE_A5, 1000},
    {NOTE_B5, 1000},
    {NOTE_C6, 1000},
    {NOTE_D6, 1000},
    {NOTE_E6, 1000},
    {NOTE_F6, 1000},
    {NOTE_G6, 1000}
};

const int songLength = sizeof(song) / sizeof(song[0]);
int currentNote = 0;

ServoState Servos[6];

bool playing = true;
unsigned long noteStartTime = 0;
unsigned long lastServoUpdate = 0;

void setup() {
    delay(5000);
    Serial.begin(921600);
    Serial.println("Entering setup");

    ledcAttachChannel(HOLE01, freq, resolution, PWMCHANNEL0);
    ledcAttachChannel(HOLE02, freq, resolution, PWMCHANNEL1);
    ledcAttachChannel(HOLE2, freq, resolution, PWMCHANNEL2);
    ledcAttachChannel(HOLE3, freq, resolution, PWMCHANNEL3);
    ledcAttachChannel(HOLE4, freq, resolution, PWMCHANNEL4);
    ledcAttachChannel(HOLE5, freq, resolution, PWMCHANNEL5);
}

```

```

    ledcWrite(HOLE01, 220);
    ledcWrite(HOLE02, 120);
    ledcWrite(HOLE2, 200);
    ledcWrite(HOLE3, 230);
    ledcWrite(HOLE4, 200);
    ledcWrite(HOLE5, 325);

    Servos[0] = {HOLE01, 220, 220, 1.0};
    Servos[1] = {HOLE02, 120, 120, 1.0};
    Servos[2] = {HOLE2, 200, 200, 1.0};
    Servos[3] = {HOLE3, 230, 230, 1.0};
    Servos[4] = {HOLE4, 200, 200, 1.0};
    Servos[5] = {HOLE5, 325, 325, 1.0};

    pinMode(HOLE1, OUTPUT);
    pinMode(HOLE6, OUTPUT);
    pinMode(HOLE7, OUTPUT);

    delay(1000);
    Serial.println("code started");
}

void loop() {
    if (millis() - lastServoUpdate >= 10) {
        updateServo(Servos[0]);
        updateServo(Servos[1]);
        updateServo(Servos[2]);
        updateServo(Servos[3]);
        updateServo(Servos[4]);
        updateServo(Servos[5]);
        lastServoUpdate = millis();
    }
    playSong();
}

void playSong() {
    if (!playing) return;

    if (millis() - noteStartTime > song[currentNote].duration) {
        noteStartTime = millis();
    }
}

```

```

    Pitch curPitch = song[currentNote].notePitch;

    moveServo(0, curPitch.hole01, 150);
    moveServo(1, curPitch.hole02, 150);
    moveServo(2, curPitch.hole2, 150);
    moveServo(3, curPitch.hole3, 150);
    moveServo(4, curPitch.hole4, 150);
    moveServo(5, curPitch.hole5, 150);

    digitalWrite(HOLE1, curPitch.hole1);
    digitalWrite(HOLE6, curPitch.hole6);
    digitalWrite(HOLE7, curPitch.hole7);

    currentNote++;
    if (currentNote >= songLength) {
        playing = false;
        currentNote = 0;
    }
}

void moveServo(int servoNumber, int target, int movementTime) {
    int targetClamped = constrain(target, min_counts, max_counts);
    Servos[servoNumber].targetPos = targetClamped;

    float distance = abs(targetClamped - Servos[servoNumber].currentPos);
    int updateTimes = movementTime / 10;
    Servos[servoNumber].speed = distance / updateTimes;

    if (Servos[servoNumber].speed < 0.2)
        Servos[servoNumber].speed = 0.2;
}

void updateServo(ServoState &servo) {
    if (servo.currentPos == servo.targetPos) return;

    if (servo.currentPos < servo.targetPos) {
        servo.currentPos += servo.speed;
        if (servo.currentPos > servo.targetPos) servo.currentPos =
servo.targetPos;

```

```
    } else {  
        servo.currentPos -= servo.speed;  
        if (servo.currentPos < servo.targetPos) servo.currentPos =  
servo.targetPos;  
    }  
    ledcWrite(servo.pin, (int)servo.currentPos);  
}
```

Potential Final

```
#include <WiFi.h>
#include "MidiData.h"

MidiData midi("192.168.8.173",5000);

const char* ssid = " ";
const char* password = " ";

const unsigned int localPort = 2808;
WiFiUDP Udp;

char packetBuffer[255];
char replyBuffer[] = "Message Received!";

int DownloadStatus = 0;

// #define MUTE1 7
// #define MUTE2 1
#define HOLE01 0
#define HOLE02 1
#define HOLE1 4
#define HOLE2 5
#define HOLE3 6
#define HOLE4 7
#define HOLE5 10
#define HOLE6 18
#define HOLE7 19

#define freq 50
#define resolution 12
#define PWMCHANNEL0 0
#define PWMCHANNEL1 1
#define PWMCHANNEL2 2
#define PWMCHANNEL3 3
#define PWMCHANNEL4 4
```

```
#define PWMCHANNEL5 5
```

```
#define closed01 205
```

```
#define open01 220
```

```
#define closed02 105
```

```
#define open02 120
```

```
#define closed2 215
```

```
#define open2 200
```

```
#define closed3 235
```

```
#define open3 230
```

```
#define closed4 235
```

```
#define open4 200
```

```
#define closed5 265
```

```
#define open5 240
```

```
int min_counts = 102;
```

```
int max_counts = 501;
```

```
struct ServoState {
```

```
    int pin;
```

```
    float currentPos;
```

```
    float targetPos;
```

```
    float speed;
```

```
};
```

```
struct Pitch {
```

```
    int hole01;
```

```
    int hole02;
```

```
    int hole1;
```

```
    int hole2;
```

```
    int hole3;
```

```
    int hole4;
```

```
    int hole5;
```

```
    int hole6;
```

```
    int hole7;
```

```
};
```

```

//const Pitch NOTE_G5
{closed01,closed02,0,closed2,closed3,open4,open5,0,0}; //79, 13V
const Pitch NOTE_A5 {closed01,closed02,0,closed2,open3,open4,open5,0,0};
// MIDI 81, 24V
const Pitch NOTE_B5 {closed01,closed02,0,open2,open3,open4,open5,0,0};
//MIDI 83,24V
const Pitch NOTE_C6
{closed01,closed02,0,closed2,closed3,closed4,closed5,1,1}; // MIDI84,24V
const Pitch NOTE_D6
{closed01,closed02,0,closed2,closed3,closed4,closed5,1,0}; // MIDI86,24V
const Pitch NOTE_E6
{closed01,closed02,0,closed2,closed3,closed4,closed5,0,0}; // MIDI88,24V
const Pitch NOTE_F6
{closed01,closed02,0,closed2,closed3,closed4,open5,0,0}; // MIDI89,24V
const Pitch NOTE_G6 {closed01,closed02,0,closed2,closed3,open4,open5,0,0};
// MIDI91,24V
const Pitch NOTE_A6
{closed01,closed02,0,closed2,open3,closed4,closed5,1,0}; // MIDI93,24V
const Pitch NONE {open01,open02,0,open2,open3,open4,open5,0,0};

int NOTE_COUNT = 0;

int *Midi_Note = nullptr;
int *start_time = nullptr;
int *noteDuration = nullptr;

uint32_t timebase;
int currentNote = 0;
bool noteActive = false;

ServoState Servos[6];
unsigned long lastServoUpdate = 0;

void moveServo(int servoNumber, int target, int movementTime) {
    int targetClamped = constrain(target, min_counts, max_counts);
    Servos[servoNumber].targetPos = targetClamped;

    float distance = abs(targetClamped - Servos[servoNumber].currentPos);
    int updateTimes = movementTime / 10;

```

```

Servos[servoNumber].speed = distance / updateTimes;

if (Servos[servoNumber].speed < 0.2)
    Servos[servoNumber].speed = 0.2;
}

void updateServo(ServoState &servo) {
    if (servo.currentPos == servo.targetPos) return;

    if (servo.currentPos < servo.targetPos) {
        servo.currentPos += servo.speed;
        if (servo.currentPos > servo.targetPos) servo.currentPos =
servo.targetPos;
    } else {
        servo.currentPos -= servo.speed;
        if (servo.currentPos < servo.targetPos) servo.currentPos =
servo.targetPos;
    }
    ledcWrite(servo.pin, (int)servo.currentPos);
}

Pitch midiToPitch(int midiNum) {
    switch(midiNum) {
        // case 72: return NOTE_C5;
        // case 74: return NOTE_D5;
        // case 76: return NOTE_E5;
        // case 77: return NOTE_F5;
        // case 79: return NOTE_G5;
        case 81: return NOTE_A5;
        case 83: return NOTE_B5;
        case 84: return NOTE_C6;
        case 86: return NOTE_D6;
        case 88: return NOTE_E6;
        case 89: return NOTE_F6;
        case 91: return NOTE_G6;
        case 93: return NOTE_A6;

        default: return NONE;
    }
}

```

```

}
}

void playMidiSong() {
    float songTimeSec = millis() - timebase;

    if(noteActive && songTimeSec >= (start_time[currentNote -
1]+noteDuration[currentNote - 1])) {
//      if (songTimeSec - (start_time[currentNote -
1]+noteDuration[currentNote - 1]) < 100) {
//          digitalWrite(MUTE1,HIGH);
//      }

    Pitch mutePitch = NONE;
    moveServo(0, mutePitch.hole01, 150);
    moveServo(1, mutePitch.hole02, 150);
    moveServo(2, mutePitch.hole2, 150);
    moveServo(3, mutePitch.hole3, 150);
    moveServo(4, mutePitch.hole4, 150);
    moveServo(5, mutePitch.hole5, 150);
    digitalWrite(HOLE1, mutePitch.hole1);
    digitalWrite(HOLE6, mutePitch.hole6);
    digitalWrite(HOLE7, mutePitch.hole7);
    noteActive = false;
}

if(currentNote < NOTE_COUNT && songTimeSec >= start_time[currentNote]) {
    Pitch curPitch = midiToPitch(Midi_Note[currentNote]);
    //digitalWrite(MUTE1,LOW);

    moveServo(0, curPitch.hole01, 150);
    moveServo(1, curPitch.hole02, 150);
    moveServo(2, curPitch.hole2, 150);
    moveServo(3, curPitch.hole3, 150);
    moveServo(4, curPitch.hole4, 150);
    moveServo(5, curPitch.hole5, 150);

    digitalWrite(HOLE1, curPitch.hole1);

```

```

    digitalWrite(HOLE6, curPitch.hole6);
    digitalWrite(HOLE7, curPitch.hole7);

    noteActive = true;
    currentNote++;
}

if(currentNote >= NOTE_COUNT && !noteActive) {
    currentNote = 0;
    timebase = millis();
}
}

void setup() {
    Serial.begin(921600);

    ledcAttachChannel(HOLE01, freq, resolution, PWMCHANNEL0);
    ledcAttachChannel(HOLE02, freq, resolution, PWMCHANNEL1);
    ledcAttachChannel(HOLE2, freq, resolution, PWMCHANNEL2);
    ledcAttachChannel(HOLE3, freq, resolution, PWMCHANNEL3);
    ledcAttachChannel(HOLE4, freq, resolution, PWMCHANNEL4);
    ledcAttachChannel(HOLE5, freq, resolution, PWMCHANNEL5);

    pinMode(HOLE1, OUTPUT);
    pinMode(HOLE6, OUTPUT);
    pinMode(HOLE7, OUTPUT);
    // pinMode(MUTE1, OUTPUT);

    ledcWrite(HOLE01, open01);
    ledcWrite(HOLE02, open02);
    ledcWrite(HOLE2, open2);
    ledcWrite(HOLE3, open3);
    ledcWrite(HOLE4, open4);
    ledcWrite(HOLE5, open5);

```

```

digitalWrite(HOLE1,0);
digitalWrite(HOLE6,0);
digitalWrite(HOLE7,0);
// digitalWrite(MUTE1,0);
// digitalWrite(MUTE2,0);


Servos[0] = {HOLE01, open01, open01, 1.0};
Servos[1] = {HOLE02, open02, open02, 1.0};
Servos[2] = {HOLE2, open2, open2, 1.0};
Servos[3] = {HOLE3, open3, open3, 1.0};
Servos[4] = {HOLE4, open4, open4, 1.0};
Servos[5] = {HOLE5, open5, open5, 1.0};


delay(1000);
timebase = millis();
}

void loop() {
  if (DownloadStatus == 0) {
    WiFi.begin(" ", " ");
    while (WiFi.status() != WL_CONNECTED) { Serial.print("."); delay(500);
  }

  Serial.printf("Trying 192.168.8.173:5000");
  midi.fetchByTrackName("Song.mid", "Recorder");
  NOTE_COUNT = midi.getNoteCount();


  Midi_Note = new int[NOTE_COUNT];
  start_time = new int[NOTE_COUNT];
  noteDuration = new int[NOTE_COUNT];


  midi.printSummary();
  size_t i;
  for (int i = 0; i < midi.getNoteCount(); i++) {
    const NoteEvent *note = midi.getNote(i);

```

```

    Midi_Note[i] = note->pitch;
    start_time[i] = note->startTimeMs;
    noteDuration[i] = note->durationMs;

    char noteBuf[6];
    midi.getNoteName(note->pitch, noteBuf, sizeof(noteBuf));
    Serial.printf("Processing Note #%d -> Pitch: %d %s at %d ms\n",
                  i, note->pitch, noteBuf, note->startTimeMs);
}
Serial.printf("%d / %d notes read\n", i, midi.getNoteCount());
DownloadStatus = 1;
}

if (DownloadStatus == 1) {
    WiFi.begin(ssid, password);
    WiFi.config(
        IPAddress(192, 168, 0, 79),
        IPAddress(192, 168, 1, 1),
        IPAddress(255, 255, 255, 0));
    while (WiFi.status() != WL_CONNECTED) {
        delay(500);
        Serial.print(".");
    }

    Serial.println("\nWiFi connected.");
    Serial.print("IP address: ");
    Serial.println(WiFi.localIP());

    Udp.begin(localPort);
    Serial.print("Listening on UDP port ");
    Serial.println(localPort);
    DownloadStatus = 2;
}
int packetSize = Udp.parsePacket();
if (packetSize) {
    int len = Udp.read(packetBuffer, 255);

```

```

    if (len > 0) {
        packetBuffer[len] = 0;
    }
    // Serial.println(packetBuffer);
    if (packetBuffer[0] == '0') {}
    //test sync "T"
    else if (packetBuffer[0] == 'T') {
        Serial.println("TEST");
        playOnenotelsec();
    }
    // Go message "G"
    else if (packetBuffer[0] == 'G') {
        Serial.println("START");
        startSong();
    }
}
}

void startSong() {
    if (millis() - lastServoUpdate >= 10) {
        updateServo(Servos[0]);
        updateServo(Servos[1]);
        updateServo(Servos[2]);
        updateServo(Servos[3]);
        updateServo(Servos[4]);
        updateServo(Servos[5]);
        lastServoUpdate = millis();
    }
    playMidiSong();
}

void playOnenotelsec() {
    Pitch curPitch = midiToPitch(Midi_Note[currentNote]);

    moveServo(0, curPitch.hole01, 150);
    moveServo(1, curPitch.hole02, 150);
    moveServo(2, curPitch.hole2, 150);

```

```
moveServo(3, curPitch.hole3, 150);  
moveServo(4, curPitch.hole4, 150);  
moveServo(5, curPitch.hole5, 150);  
  
digitalWrite(HOLE1, curPitch.hole1);  
digitalWrite(HOLE6, curPitch.hole6);  
digitalWrite(HOLE7, curPitch.hole7);  
}
```

Def Works

```
#include <WiFi.h>
#include "MidiData.h"

#define HOLE01 0
#define HOLE02 1
#define HOLE1 4
#define HOLE2 5
#define HOLE3 6
#define HOLE4 7
#define HOLE5 10
#define HOLE6 18
#define HOLE7 19

#define freq 50
#define resolution 12
#define PWMCHANNEL0 0
#define PWMCHANNEL1 1
#define PWMCHANNEL2 2
#define PWMCHANNEL3 3
#define PWMCHANNEL4 4
#define PWMCHANNEL5 5

#define closed01 205
#define open01 220
#define closed02 300
#define open02 280
#define closed2 225
#define open2 200
#define closed3 235
#define open3 230
#define closed4 235
#define open4 200
#define closed5 265
#define open5 240

int min_counts = 102;
int max_counts = 501;
```

```

struct ServoState {
    int pin;
    float currentPos;
    float targetPos;
    float speed;
};

struct Pitch {
    int hole01;
    int hole02;
    int hole1;
    int hole2;
    int hole3;
    int hole4;
    int hole5;
    int hole6;
    int hole7;
};

const Pitch NOTE_A5 {closed01,closed02,0,closed2,open3,open4,open5,0,0};
// MIDI 81
const Pitch NOTE_B5 {closed01,closed02,0,open2,open3,open4,open5,0,0}; //
MIDI 83
const Pitch NOTE_C6 {closed01,closed02,1,closed2,open3,open4,open5,0,0};
// MIDI84
const Pitch NOTE_D6
{closed01,closed02,0,closed2,closed3,closed4,closed5,1,0}; // MIDI86
const Pitch NOTE_E6
{closed01,closed02,0,closed2,closed3,closed4,closed5,0,0}; // MIDI88
const Pitch NOTE_F6
{closed01,closed02,0,closed2,closed3,closed4,open5,0,0}; // MIDI89
const Pitch NOTE_G6 {closed01,closed02,0,closed2,closed3,open4,open5,0,0};
// MIDI91
const Pitch NOTE_A6
{closed01,closed02,0,closed2,open3,closed4,closed5,1,0}; // MIDI93
const Pitch NONE {open01,open02,0,open2,open3,open4,open5,0,0};

```

```
MidiData midi("192.168.8.173", 5000);

int NOTE_COUNT = 0;

int *Midi_Note = nullptr;
int *start_time = nullptr;
int *noteDuration = nullptr;

const char* ssid = " ";
const char* password = " ";

const unsigned int localPort = 2808;
WiFiUDP Udp;

char packetBuffer[255];
char replyBuffer[] = "Message Received!";

int DownloadStatus = 0;

uint32_t timebase;
int currentNote = 0;
bool noteActive = false;

ServoState Servos[6];
unsigned long lastServoUpdate = 0;

void moveServo(int servoNumber, int target, int movementTime) {
    int targetClamped = constrain(target, min_counts, max_counts);
    Servos[servoNumber].targetPos = targetClamped;

    float distance = abs(targetClamped - Servos[servoNumber].currentPos);
    int updateTimes = movementTime / 10;
    Servos[servoNumber].speed = distance / updateTimes;

    if (Servos[servoNumber].speed < 0.2)
        Servos[servoNumber].speed = 0.2;
}
```

```

void updateServo(ServoState &servo) {
    if (servo.currentPos == servo.targetPos) return;

    if (servo.currentPos < servo.targetPos) {
        servo.currentPos += servo.speed;
        if (servo.currentPos > servo.targetPos) servo.currentPos =
servo.targetPos;
    } else {
        servo.currentPos -= servo.speed;
        if (servo.currentPos < servo.targetPos) servo.currentPos =
servo.targetPos;
    }
    ledcWrite(servo.pin, (int)servo.currentPos);
}

```

```

Pitch midiToPitch(int midiNum) {
    switch(midiNum) {
        // case 72: return NOTE_C5;
        // case 74: return NOTE_D5;
        // case 76: return NOTE_E5;
        // case 77: return NOTE_F5;
        // case 79: return NOTE_G5;
        case 81: return NOTE_A5;
        case 83: return NOTE_B5;
        case 84: return NOTE_C6;
        case 86: return NOTE_D6;
        case 88: return NOTE_E6;
        case 89: return NOTE_F6;
        case 91: return NOTE_G6;
        case 93: return NOTE_A6;

        default: return NONE;
    }
}

```

```

void playMidiSong() {
    float songTimeSec = millis() - timebase;

```

```

    if(noteActive && songTimeSec >= (start_time[currentNote -
1]+noteDuration[currentNote - 1])) {
//      if (songTimeSec - (start_time[currentNote -
1]+noteDuration[currentNote - 1]) > 100) {
//          digitalWrite(MUTE1,HIGH);
//      } else {
//          digitalWrite(MUTE1,HIGH);
//      }

    Pitch mutePitch = NONE;
    moveServo(0, mutePitch.hole01, 150);
    moveServo(1, mutePitch.hole02, 150);
    moveServo(2, mutePitch.hole2, 150);
    moveServo(3, mutePitch.hole3, 150);
    moveServo(4, mutePitch.hole4, 150);
    moveServo(5, mutePitch.hole5, 150);
    digitalWrite(HOLE1, mutePitch.hole1);
    digitalWrite(HOLE6, mutePitch.hole6);
    digitalWrite(HOLE7, mutePitch.hole7);
    noteActive = false;
}

if(currentNote < NOTE_COUNT && songTimeSec >= start_time[currentNote]) {
    Pitch curPitch = midiToPitch(Midi_Note[currentNote]);
    moveServo(0, curPitch.hole01, 150);
    moveServo(1, curPitch.hole02, 150);
    moveServo(2, curPitch.hole2, 150);
    moveServo(3, curPitch.hole3, 150);
    moveServo(4, curPitch.hole4, 150);
    moveServo(5, curPitch.hole5, 150);

    digitalWrite(HOLE1, curPitch.hole1);
    digitalWrite(HOLE6, curPitch.hole6);
    digitalWrite(HOLE7, curPitch.hole7);

    noteActive = true;
    currentNote++;
}

```

```

if(currentNote >= NOTE_COUNT && !noteActive) {
    currentNote = 0;
    timebase = millis();
}
}

void setup() {
    Serial.begin(921600);
    ledcAttachChannel(HOLE01, freq, resolution, PWMCHANNEL0);
    ledcAttachChannel(HOLE02, freq, resolution, PWMCHANNEL1);
    ledcAttachChannel(HOLE2, freq, resolution, PWMCHANNEL2);
    ledcAttachChannel(HOLE3, freq, resolution, PWMCHANNEL3);
    ledcAttachChannel(HOLE4, freq, resolution, PWMCHANNEL4);
    ledcAttachChannel(HOLE5, freq, resolution, PWMCHANNEL5);

    pinMode(HOLE1, OUTPUT);
    pinMode(HOLE6, OUTPUT);
    pinMode(HOLE7, OUTPUT);
    // pinMode(MUTE1, OUTPUT);

    // ledcWrite(HOLE01, open01);
    // ledcWrite(HOLE02, open02);
    // ledcWrite(HOLE2, open2);
    // ledcWrite(HOLE3, open3);
    // ledcWrite(HOLE4, open4);
    // ledcWrite(HOLE5, open5);
    // digitalWrite(HOLE1,0);
    // digitalWrite(HOLE6,0);
    // digitalWrite(HOLE7,0);
    // digitalWrite(MUTE1,0);

    Servos[0] = {HOLE01, open01, open01, 1.0};
}

```

```

Servos[1] = {HOLE02, open02, open02, 1.0};
Servos[2] = {HOLE2, open2, open2, 1.0};
Servos[3] = {HOLE3, open3, open3, 1.0};
Servos[4] = {HOLE4, open4, open4, 1.0};
Servos[5] = {HOLE5, open5, open5, 1.0};

timebase = millis();

}

void loop() {

    if (DownloadStatus == 0) {
        /*
        // OPTION A: Good for debugging hardware without requiring wifi:
        // Pass array for testing the hardware (no wifi required)
        // Define static array for testing notes
        const NoteEvent myTestNotes[] = {
            {60, 0, 400, 100}, // C4, starttime, duration, velocity -- time
in ms, velocity usually not used.
            {62, 500, 400, 100}, // D4
            {64, 1000, 400, 100}, // E4
            {65, 1500, 400, 100}, // F4
            {67, 2000, 800, 100} // G4
        };

        size_t totalNotes = sizeof(myTestNotes) / sizeof(NoteEvent);
        midi.setNotes(myTestNotes, totalNotes);
        */

        // OPTION B: Connect to Wi-Fi and fetch songs from server
        WiFi.begin(" ", " ");
        while (WiFi.status() != WL_CONNECTED) { Serial.print(".");
delay(500); }
        Serial.printf("Trying 192.168.8.173:5000");
    }
}

```

```

        midi.fetchByTrackName("Song.mid", "Recorder");  /***Update with
your Instrument***/

        // Custom loop logic

        // Example accessing note pitch and start time.
        midi.printSummary(); // print first 10 notes just to see what they
are

        size_t i;
        for (i = 0; i < midi.getNoteCount(); i++) {
            const NoteEvent* note = midi.getNote(i);
            char noteBuf[6];
            // Midi_Note[i] = note->pitch;
            // start_time[i] = note->startTimeMs;
            // noteDuration[i] = note->durationMs;

            midi.getNoteName(note->pitch, noteBuf, sizeof(noteBuf));
            Serial.printf("Processing Note #%d -> Pitch: %d %s at %d
ms\n",
                        i, note->pitch, noteBuf, note->startTimeMs);
        }
        Serial.printf("%d / %d notes read\n", i, midi.getNoteCount());
        DownloadStatus = 1;
    }

    if (DownloadStatus == 1) {
        WiFi.begin(ssid, password);

        WiFi.config(
            IPAddress(192, 168, 0, 79),
            IPAddress(192, 168, 1, 1),
            IPAddress(255, 255, 255, 0));

        while (WiFi.status() != WL_CONNECTED) {
            delay(500);
            Serial.print(".");

```

```

    }
    Serial.println("\nWiFi connected.");
    Serial.print("IP address: ");
    Serial.println(WiFi.localIP());

    Udp.begin(localPort);
    Serial.print("Listening on UDP port ");
    Serial.println(localPort);
    DownloadStatus = 2;
}

int packetSize = Udp.parsePacket();

if (packetSize) {
    int len = Udp.read(packetBuffer, 255);
    if (len > 0) {
        packetBuffer[len] = 0;
    }
    // Serial.println(packetBuffer);
    if (packetBuffer[0] == '0') {}
    // test sync "T"
    else if (packetBuffer[0] == 'T') {
        Serial.println("TEST");
        //playOnenotelsec();
    }
    else if (packetBuffer[0] == 'G') {
        Serial.println("START");
        //startSong();
    }
}

}

void startSong() {
    if (millis() - lastServoUpdate >= 10) {
        updateServo(Servos[0]);
        updateServo(Servos[1]);
        updateServo(Servos[2]);
        updateServo(Servos[3]);
        updateServo(Servos[4]);
    }
}

```

```
    updateServo(Servos[5]);  
    lastServoUpdate = millis();  
}  
playMidiSong();  
}  
  
void playOnenotelsec() {  
    Pitch curPitch = midiToPitch(Midi_Note[currentNote]);  
  
    moveServo(0, curPitch.hole01, 150);  
    moveServo(1, curPitch.hole02, 150);  
    moveServo(2, curPitch.hole2, 150);  
    moveServo(3, curPitch.hole3, 150);  
    moveServo(4, curPitch.hole4, 150);  
    moveServo(5, curPitch.hole5, 150);  
  
    digitalWrite(HOLE1, curPitch.hole1);  
    digitalWrite(HOLE6, curPitch.hole6);  
    digitalWrite(HOLE7, curPitch.hole7);  
}
```

MUTE CODE

```
#include <WiFi.h>
#include "MidiData.h"

MidiData midi("192.168.8.173",5000);

const char* ssid = " ";
const char* password = " ";

// UDP Config
const unsigned int localPort = 2808;
WiFiUDP Udp;

char packetBuffer[255];
char replyBuffer[] = "Message Received!";

int DownloadStatus = 0;

#define MUTE1 4
#define MUTE2 5

#define freq 50
#define resolution 12
#define PWMCHANNEL0 0

#define openM 220
#define closedM 240
#define open01 200
#define closed01 200
#define open02 200
#define closed02 200
#define open2 200
#define closed2 200
#define open3 200
#define closed3 200
#define open4 200
#define closed4 200
#define open5 200
#define closed5 200

int min_counts = 102;
```

```

int max_counts = 501;

struct ServoState {
int pin;
float currentPos;
float targetPos;
float speed;
};

struct Pitch {
int hole01;
int hole02;
int hole1;
int hole2;
int hole3;
int hole4;
int hole5;
int hole6;
int hole7;
int mute;
};

//const Pitch NOTE_G5 {closed01,closed02,0,closed2,closed3,open4,open5,0,0}; //79, 13V
const Pitch NOTE_A5 {closed01,closed02,0,closed2,open3,open4,open5,0,0,0}; // MIDI 81,
24V
const Pitch NOTE_B5 {closed01,closed02,0,open2,open3,open4,open5,0,0,0}; //MIDI 83,24V
const Pitch NOTE_C6 {closed01,closed02,0,closed2,closed3,closed4,closed5,1,1,0}; //
MIDI84,24V
const Pitch NOTE_D6 {closed01,closed02,0,closed2,closed3,closed4,closed5,1,0,0}; //
MIDI86,24V
const Pitch NOTE_E6 {closed01,closed02,0,closed2,closed3,closed4,closed5,0,0,0}; //
MIDI88,24V
const Pitch NOTE_F6 {closed01,closed02,0,closed2,closed3,closed4,open5,0,0,0}; //
MIDI89,24V
const Pitch NOTE_G6 {closed01,closed02,0,closed2,closed3,open4,open5,0,0,0}; //
MIDI91,24V
const Pitch NOTE_A6 {closed01,closed02,0,closed2,open3,closed4,closed5,1,0,0}; //
MIDI93,24V
const Pitch NONE {open01,open02,0,open2,open3,open4,open5,0,0,1};

int NOTE_COUNT = 0;

```

```

int *Midi_Note = nullptr;
int *start_time = nullptr;
int *noteDuration = nullptr;

uint32_t timebase;
int currentNote = 0;
bool noteActive = false;

ServoState Servos[1];
unsigned long lastServoUpdate = 0;

void moveServo(int servoNumber, int target, int movementTime) {
    int targetClamped = constrain(target, min_counts, max_counts);
    Servos[servoNumber].targetPos = targetClamped;

    float distance = abs(targetClamped - Servos[servoNumber].currentPos);
    int updateTimes = movementTime / 10;
    Servos[servoNumber].speed = distance / updateTimes;

    if (Servos[servoNumber].speed < 0.2)
        Servos[servoNumber].speed = 0.2;
}

void updateServo(ServoState &servo) {
    if (servo.currentPos == servo.targetPos) return;

    if (servo.currentPos < servo.targetPos) {
        servo.currentPos += servo.speed;
        if (servo.currentPos > servo.targetPos) servo.currentPos = servo.targetPos;
    } else {
        servo.currentPos -= servo.speed;
        if (servo.currentPos < servo.targetPos) servo.currentPos = servo.targetPos;
    }

    ledcWrite(servo.pin, (int)servo.currentPos);
}

Pitch midiToPitch(int midiNum) {
    switch(midiNum) {
        case 72: return NOTE_C6;
        case 74: return NOTE_D6;
        case 76: return NOTE_E6;
    }
}

```

```

    case 77: return NOTE_F6;
    case 79: return NOTE_G6;
    case 81: return NOTE_A6;
    case 83: return NOTE_B5;
    case 84: return NOTE_C6;
    case 86: return NOTE_D6;
    case 88: return NOTE_E6;
    case 89: return NOTE_F6;
    case 91: return NOTE_G6;
    case 93: return NOTE_A6;
    case 96: return NOTE_C6;
    case 98: return NOTE_D6;
    case 100: return NOTE_E6;
    case 101: return NOTE_F6;
    case 103: return NOTE_G6;
    case 105: return NOTE_A6;
    default: return NONE;
}
}

void playMidiSong() {
float songTimeSec = millis() - timebase;

    if(noteActive && songTimeSec >= (start_time[currentNote -
1]+noteDuration[currentNote - 1])) {
        if (songTimeSec - (start_time[currentNote - 1]+noteDuration[currentNote - 1]) >
100) {
            moveServo(0, closedM, 150);
        } else {
            digitalWrite(MUTE1,HIGH);
        }
        noteActive = false;
    }

    if(currentNote < NOTE_COUNT && songTimeSec >= start_time[currentNote]) {
        moveServo(0, openM, 150);
        digitalWrite(MUTE1,LOW);
        noteActive = true;
        currentNote++;
    }

    if(currentNote >= NOTE_COUNT && !noteActive) {

```

```

    currentNote = 0;
    timebase = millis();
}

}

void setup() {
    Serial.begin(115200);
    ledcAttachChannel(MUTE2, freq, resolution, PWMCHANNEL0);
    ledcWrite(MUTE2, open01);
    Servos[0] = {MUTE2, open01, open01, 1.0};
    pinMode(MUTE1, OUTPUT);
    digitalWrite(MUTE1, LOW);
    delay(1000);
    timebase = millis();
}

void loop() {
    if (DownloadStatus == 0) {
        WiFi.begin(" ", " ");
        while (WiFi.status() != WL_CONNECTED) { Serial.print("."); delay(500); }
        Serial.printf("Trying 192.168.8.173:5000");
        midi.fetchByTrackName("Song.mid", "Recorder");
        NOTE_COUNT = midi.getNoteCount();

        Midi_Note = new int[NOTE_COUNT];
        start_time = new int[NOTE_COUNT];
        noteDuration = new int[NOTE_COUNT];

        midi.printSummary();
        size_t i;
        for (int i = 0; i < midi.getNoteCount(); i++) {
            const NoteEvent *note = midi.getNote(i);
            Midi_Note[i] = note->pitch;
            start_time[i] = note->startTimeMs;
            noteDuration[i] = note->durationMs;

            char noteBuf[6];
            midi.getNoteName(note->pitch, noteBuf, sizeof(noteBuf));

```

```

        Serial.printf("Processing Note #%d -> Pitch: %d %s at %d ms\n",
                        i, note->pitch, noteBuf, note->startTimeMs);
    }
    Serial.printf("%d / %d notes read\n", i, midi.getNoteCount());
    DownloadStatus = 1;
}

if (DownloadStatus == 1) {
    WiFi.begin(ssid, password);
    WiFi.config(
        IPAddress(192, 168, 0, 78),
        IPAddress(192, 168, 1, 1),
        IPAddress(255, 255, 255, 0));
    while (WiFi.status() != WL_CONNECTED) {
        delay(500);
        Serial.print(".");
    }

    Serial.println("\nWiFi connected.");
    Serial.print("IP address: ");
    Serial.println(WiFi.localIP());

    Udp.begin(localPort);
    Serial.print("Listening on UDP port ");
    Serial.println(localPort);
    DownloadStatus = 2;
}

int packetSize = Udp.parsePacket();
if (packetSize) {
    int len = Udp.read(packetBuffer, 255);
    if (len > 0) {
        packetBuffer[len] = 0;
    }
    // Serial.println(packetBuffer);
    if (packetBuffer[0] == '0') {}
    else if (packetBuffer[0] == 'T') {
        Serial.println("TEST");
        playOnenotelsec();
    }
}

```

```
else if (packetBuffer[0] == 'G') {  
    Serial.println("START");  
    startSong();  
}  
}  
}  
  
void startSong() {  
if (millis() - lastServoUpdate >= 10) {  
    updateServo(Servos[0]);  
    lastServoUpdate = millis();  
}  
playMidiSong();  
}  
  
void playOnenote1sec() {  
while(true) {  
digitalWrite(MUTE1,HIGH);  
digitalWrite(MUTE2,LOW);  
}  
}
```