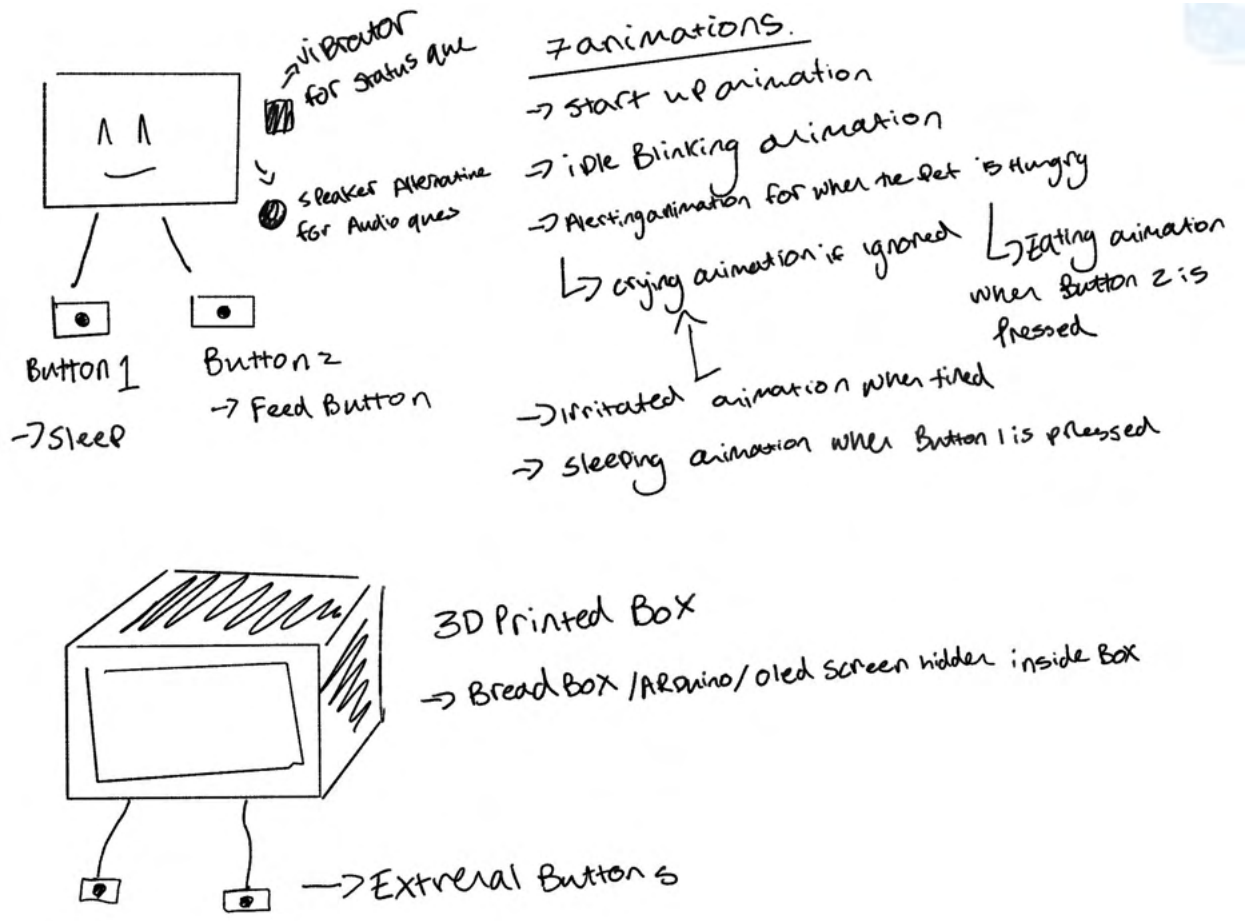
Creative Making: Experience and Physical Computing final project

Original draft



These are the animations/sprites I created for my virtual pet; there are 6 states

1) Irritated emote to indicate the want for sleep (button 1)

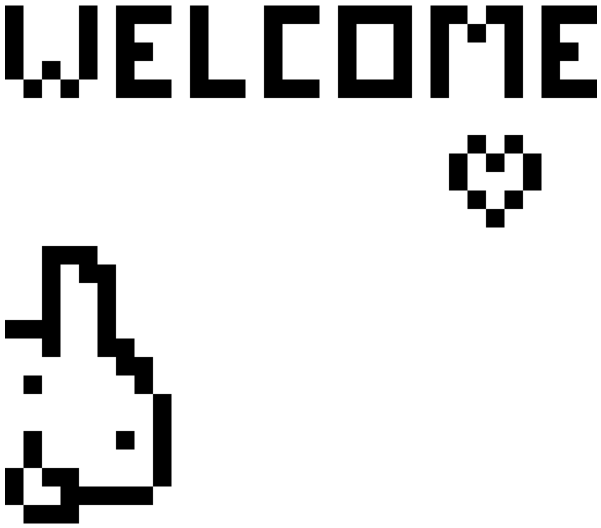
2) Eating emote triggered when button 2 is pressed

3) Sleeping emote triggered after button 1 is pressed

4) Idle blinking emote neutral for when there are no needs to be fulfilled

5) Alerting emote to indicate hunger (button 2)

6) Crying emote triggered if either irritated emote or alert emote is left up for a certain duration of time



Start up screen

Inspiration picture for my pet

In the end, I had to simplify some of the animations due to the lack of RAM space on my Arduino

in the end; the logic was as follows :

idle> alert > crying goes into mad and loops back to alert, which goes back to crying, and so on.

Pressing the sleep button will break the loop and send it into sleep animation, which will loop back to Idle and start the whole process again.

Resources I used :

https://ezgif.com/split

https://diyusthad.com/image2cpp

https://www.thingiverse.com/thing:4896543

https://www.instructables.com/Tamanooki-an-Arduino-Based-Virtual-Pet-tamagotchi-

https://www.instructables.com/Running-Animations-on-OLED-DISPLAY-SSD1306/

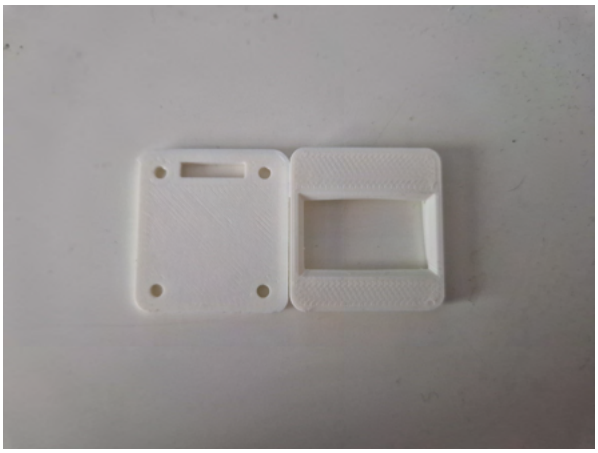
https://alojzjakob.github.io/Tamaguino/

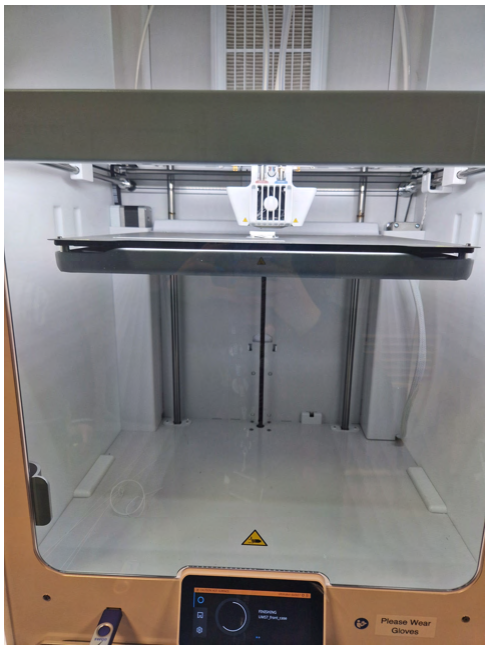
https://www.instructables.com/Create-Animation-on-OLED-Display-Controlled-by-Ard/

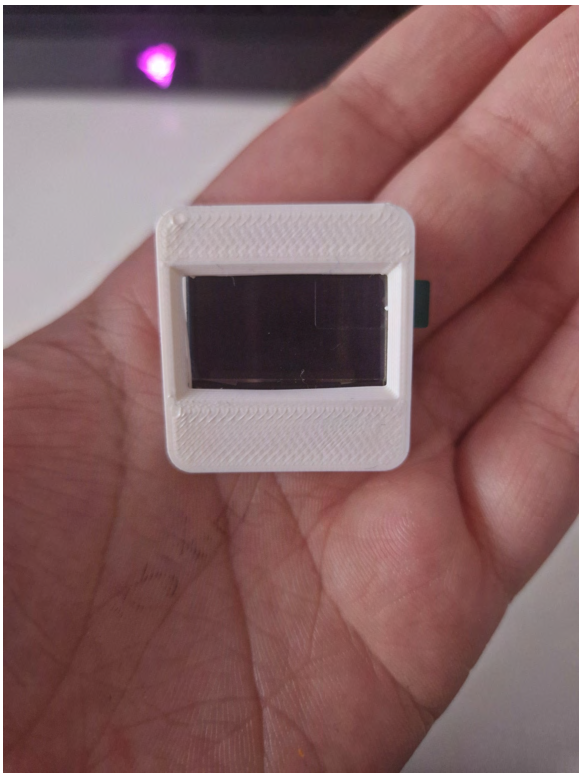
https://www.instructables.com/Create-Animation-on-OLED-Display-Controlled-by-Ard/

https://projecthub.arduino.cc/najad/interfacing-and-displaying-images-on-oled-08b4f2

The first step was 3D printing a housing for my OLED, I Used the 3D model above , filed it down and spray painted it pink for aesthetics





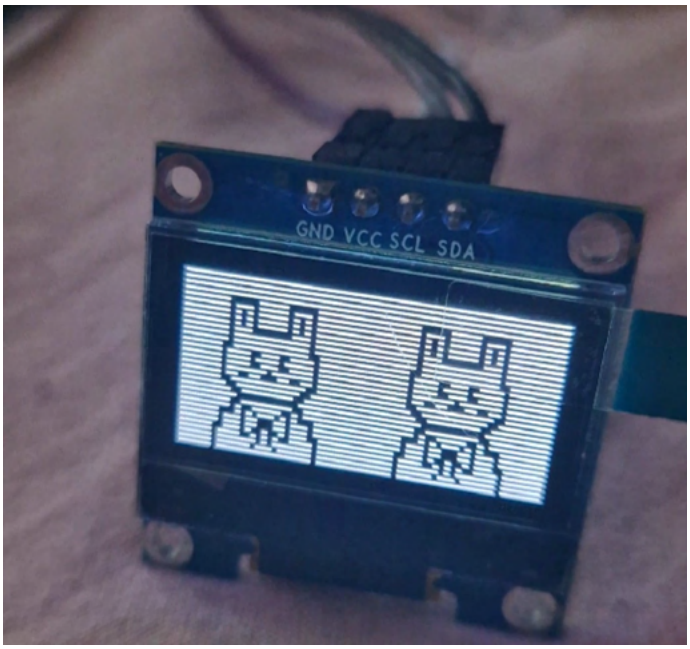


```
Sketch uses 28954 bytes (100%) of program storage space. Maximum is 28672 bytes.
Sketch too big; see https://support.arduino.cc/hc/en-us/articles/360013825179 for tips on reducing it.
Global variables use 541 bytes (21%) of dynamic memory, leaving 2019 bytes for local variables. Maximum is 2560 bytes.
text section exceeds available space in board

Compilation error: text section exceeds available space in board
```

I had to scrap the eating animation and startup animation and reduce the size of the other animations.

Another issue I faced besides storage was scaling as for reasons I could not figure out, when the dimensions of 64×64 were displayed, the image would display side by side instead of just once



I split the code up into 3 parts to insure each part would execute properly

PART 1 of the code was to test that if idle surpassed the 10 second duration that it would go into the alert status (animation)

```
#include <CuteBuzzerSounds.h>
#include <Sounds.h>

#include <SPI.h>
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>

#define SSD1306_NO_SPLASH
#define OLED_RESET 4
Adafruit_SSD1306 display(OLED_RESET);

#define SCREEN_WIDTH 128 // OLED display width, in pixels
#define SCREEN_HEIGHT 64 // OLED display height, in pixels

#define BUZZER_PIN 8
```

```
#define NUM_FRAMES_IDLE 2
#define NUM_FRAMES_ALERT 2

#define FRAME_DURATION 500
#define IDLE_DURATION 15000
#define ALERT_DURATION 10000

//idle
const unsigned char frame0 [] PROGMEM = {
ETC
};

const unsigned char frame1 [] PROGMEM = {
ETC
};

const unsigned char frame2 [] PROGMEM = {
ETC
};

const unsigned char frame3 [] PROGMEM = {
ETC};

unsigned long lastFrameTime = 0;
unsigned long lastSoundTime = 0;
bool soundPlayed = false;
unsigned long idleStartTime = 0;
unsigned long alertStartTime = 0;
bool isIdle = true;
bool isAlert = false;
int currentFrame = 0;

void setup() {
    display.begin(SSD1306_SWITCHCAPVCC, 0x3C);
    display.display();
```

```
    delay(2000);
    display.clearDisplay();
    idleStartTime = millis();
     cute.init(BUZZER_PIN);
}

void loop() {
    unsigned long currentTime = millis();
   if (!soundPlayed && currentTime - lastSoundTime >= IDLE_DU
RATION) {
        cute.play(S_CUDDLY);
        soundPlayed = true;
        lastSoundTime = currentTime;
  }
    if (isIdle && (currentTime - idleStartTime >= IDLE_DURATI
ON)) {
        // Switch to alert animation
        isIdle = false;
        isAlert = true;
        alertStartTime = currentTime;
        currentFrame = 0; // Reset frame for alert animation
    }

    if (isAlert && (currentTime - alertStartTime >= ALERT_DUR
ATION)) {
        // Stop alert animation and reset
        isAlert = false;
        idleStartTime = currentTime;
        isIdle = true;
        currentFrame = 0; // Reset frame for idle animation
    }

    if (isIdle) {
        // Idle animation
        display.clearDisplay();
        if (currentFrame == 0) {
```

```
            display.drawBitmap(0, 0, frame0, SCREEN_WIDTH, SC
REEN_HEIGHT, WHITE);
        } else {
            display.drawBitmap(0, 0, frame1, SCREEN_WIDTH, SC
REEN_HEIGHT, WHITE);
        }
        display.display();
        delay(FRAME_DURATION);
        currentFrame = (currentFrame + 1) % NUM_FRAMES_IDLE;
    } else if (isAlert) {
        // Alert animation
        display.clearDisplay();
        if (currentFrame == 0) {
            display.drawBitmap(0, 0, frame2, SCREEN_WIDTH, SC
REEN_HEIGHT, WHITE);
        } else {
            display.drawBitmap(0, 0, frame3, SCREEN_WIDTH, SC
REEN_HEIGHT, WHITE);
             cute.play(S_SURPRISE);
        }
        display.display();
        delay(FRAME_DURATION);
        currentFrame = (currentFrame + 1) % NUM_FRAMES_ALERT;
    }
}
```

now to test that idle > alert > crying

```
#include <CuteBuzzerSounds.h>
#include <Sounds.h>

#include <SPI.h>
#include <Wire.h>
#include <Adafruit_GFX.h>
```

```
#include <Adafruit_SSD1306.h>

#define SSD1306_NO_SPLASH
#define OLED_RESET 4
Adafruit_SSD1306 display(OLED_RESET);

#define SCREEN_WIDTH 128 // OLED display width, in pixels
#define SCREEN_HEIGHT 64 // OLED display height, in pixels

#define BUZZER_PIN 8

#define NUM_FRAMES_ALERT 2
#define NUM_FRAMES_CRYING 4

#define FRAME_DURATION 500
#define ALERT_DURATION 5000
#define CRYING_DURATION 10000

//alert
const unsigned char frame2 [] PROGMEM = {
ETC....
};

const unsigned char frame3 [] PROGMEM = {
ETC
};
// Define the crying frames
ETC
};
const unsigned char frame14 [] PROGMEM = {
ETC
};
const unsigned char frame15 [] PROGMEM = {
ETC
};
const unsigned char frame16 [] PROGMEM = {
```

```
ETC
};
unsigned long lastFrameTime = 0;
unsigned long lastSoundTime = 0;
bool soundPlayed = false;
unsigned long alertStartTime = 0;
unsigned long cryingStartTime = 0;
bool isAlert = true;
bool isCrying = false;
int currentFrame = 0;

void setup() {
    display.begin(SSD1306_SWITCHCAPVCC, 0x3C);
    display.display();
    delay(2000);
    display.clearDisplay();
    alertStartTime = millis();
     cute.init(BUZZER_PIN);
}
void loop() {
    unsigned long currentTime = millis();

    if (isAlert && (currentTime - alertStartTime >= ALERT_DUR
ATION)) {
        // Switch to crying animation
        isAlert = false;
        isCrying = true;
        cryingStartTime = currentTime;
        currentFrame = 0; // Reset frame for crying animation
    }

    if (isCrying && (currentTime - cryingStartTime >= CRYING_
DURATION)) {
        // Stop crying animation and reset
        isCrying = false;
        alertStartTime = currentTime;
```

```
        isAlert = true;
        currentFrame = 0; // Reset frame for alert animation
    }

    if (isAlert) {
        // Idle animation
        display.clearDisplay();
        if (currentFrame == 0) {
            display.drawBitmap(0, 0, frame2, SCREEN_WIDTH, SC
REEN_HEIGHT, WHITE);
        } else {
            display.drawBitmap(0, 0, frame3, SCREEN_WIDTH, SC
REEN_HEIGHT, WHITE);
            cute.play(S_SURPRISE);
        }
        display.display();
        delay(FRAME_DURATION);
        currentFrame = (currentFrame + 1) % NUM_FRAMES_ALERT;
    }

    if (isCrying) {
        // Crying animation
        display.clearDisplay();
        if (currentFrame == 0) {
            display.drawBitmap(0, 0, frame13, SCREEN_WIDTH, S
CREEN_HEIGHT, WHITE);
        } else if (currentFrame == 1) {
            display.drawBitmap(0, 0, frame14, SCREEN_WIDTH, S
CREEN_HEIGHT, WHITE);
        } else if (currentFrame == 2) {
            display.drawBitmap(0, 0, frame15, SCREEN_WIDTH, S
CREEN_HEIGHT, WHITE);
        } else if (currentFrame == 3) {
            display.drawBitmap(0, 0, frame16, SCREEN_WIDTH, S
CREEN_HEIGHT, WHITE);
            cute.play(S_SAD);
```

```
        }
        display.display();
        delay(FRAME_DURATION);
        currentFrame = (currentFrame + 1) % NUM_FRAMES_CRYIN
G;
    }
}
```

final part combines all the states besides the sleep state

```
#include <CuteBuzzerSounds.h>
#include <Sounds.h>

#include <SPI.h>
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>

#define SSD1306_NO_SPLASH
#define OLED_RESET 4
Adafruit_SSD1306 display(OLED_RESET);

#define SCREEN_WIDTH 128 // OLED display width, in pixels
#define SCREEN_HEIGHT 64 // OLED display height, in pixels

#define BUZZER_PIN 8
//#define SLEEP_BUTTON_PIN 2 // Assuming the button is connec
ted to pin 2
#define BUTTON_PIN 7
#define NUM_FRAMES_IDLE 2
#define NUM_FRAMES_ALERT 2
#define NUM_FRAMES_CRYING 4
#define NUM_FRAMES_MAD 4
#define NUM_FRAMES_SLEEP 4
```

```
#define FRAME_DURATION 500
#define IDLE_DURATION 8000
#define ALERT_DURATION 5000
#define CRYING_DURATION 10000
#define MAD_DURATION 10000
#define SLEEP_DURATION 10000
//idle
const unsigned char frame0 [] PROGMEM = {
etc
};

const unsigned char frame1 [] PROGMEM = {
etc
};
//alert
const unsigned char frame2 [] PROGMEM = {
etc
};

const unsigned char frame3 [] PROGMEM = {
 etc
};
// Define the crying frames
const unsigned char frame13 [] PROGMEM = {
 etc
};
const unsigned char frame14 [] PROGMEM = {
etc
};
const unsigned char frame15 [] PROGMEM = {
etc
};
const unsigned char frame16 [] PROGMEM = {
    etc
};
```

```
const unsigned char frame17 [] PROGMEM = {
etc
};
// 'frame18', 128x32px
const unsigned char frame18 [] PROGMEM = {
etc
};
// 'frame19', 128x32px
const unsigned char frame19 [] PROGMEM = {
etc
};
// 'frame20', 128x32px
const unsigned char frame20 [] PROGMEM = {
etc
};
//sleep
const unsigned char frame21 [] PROGMEM = {
    etc
};
// 'frame22', 128x32px
const unsigned char frame22 [] PROGMEM = {
etc
};
// '24', 128x32px


unsigned long lastFrameTime = 0;
unsigned long lastSoundTime = 0;
bool soundPlayed = false;
unsigned long idleStartTime = 0;
unsigned long alertStartTime = 0;
unsigned long cryingStartTime = 0;
unsigned long madStartTime = 0;
unsigned long sleepStartTime = 0;
bool isIdle = true;
bool isAlert = false;
```

```
bool isCrying = false;
bool isMad = false;
bool isSleeping = false;
int currentFrame = 0;

void setup() {
    display.begin(SSD1306_SWITCHCAPVCC, 0x3C);
    display.display();
    delay(2000);
    display.clearDisplay();
    idleStartTime = millis();
     cute.init(BUZZER_PIN);
     pinMode(BUTTON_PIN, INPUT);
}
void loop() {
    unsigned long currentTime = millis();
    int buttonState = digitalRead(BUTTON_PIN); // Read the st
ate of the button

    if (isIdle && (currentTime - idleStartTime >= IDLE_DURATI
ON)) {
        // Transition from idle to alert
        isIdle = false;
        isAlert = true;
        alertStartTime = currentTime;
        currentFrame = 0; // Reset frame for alert animation
    }

    if (isAlert && (currentTime - alertStartTime >= ALERT_DUR
ATION)) {
        // Transition from alert to crying
        isAlert = false;
        isCrying = true;
        cryingStartTime = currentTime;
        currentFrame = 0; // Reset frame for crying animation
    }
```

```
    if (isCrying && (currentTime - cryingStartTime >= CRYING_
DURATION)) {
        // Stop crying animation and transition to mad
        isCrying = false;
        isMad = true;
        madStartTime = currentTime;
        currentFrame = 0; // Reset frame for mad animation
    }

    if (isMad && (currentTime - madStartTime >= MAD_DURATIO
N)) {
        // Transition from mad back to alert
        isMad = false;
        isAlert = true;
        alertStartTime = currentTime;
        currentFrame = 0; // Reset frame for alert animation
    }

    // Display animations based on current state
    if (isIdle) {
        // Idle animation
        display.clearDisplay();
        if (currentFrame == 0) {
            display.drawBitmap(0, 0, frame0, SCREEN_WIDTH, SC
REEN_HEIGHT, WHITE);
        } else {
            display.drawBitmap(0, 0, frame1, SCREEN_WIDTH, SC
REEN_HEIGHT, WHITE);
        }
        display.display();
        delay(FRAME_DURATION);
        currentFrame = (currentFrame + 1) % NUM_FRAMES_IDLE;
    }

    if (isAlert) {
```

```
        // Alert animation
        display.clearDisplay();
        if (currentFrame == 0) {
            display.drawBitmap(0, 0, frame2, SCREEN_WIDTH, SC
REEN_HEIGHT, WHITE);
        } else {
            display.drawBitmap(0, 0, frame3, SCREEN_WIDTH, SC
REEN_HEIGHT, WHITE);
            cute.play(S_SURPRISE);
        }
        display.display();
        delay(FRAME_DURATION);
        currentFrame = (currentFrame + 1) % NUM_FRAMES_ALERT;
    }

    if (isCrying) {
        // Crying animation
        display.clearDisplay();
        if (currentFrame == 0) {
            display.drawBitmap(0, 0, frame13, SCREEN_WIDTH, S
CREEN_HEIGHT, WHITE);
        } else if (currentFrame == 1) {
            display.drawBitmap(0, 0, frame14, SCREEN_WIDTH, S
CREEN_HEIGHT, WHITE);
        } else if (currentFrame == 2) {
            display.drawBitmap(0, 0, frame15, SCREEN_WIDTH, S
CREEN_HEIGHT, WHITE);
        } else if (currentFrame == 3) {
            display.drawBitmap(0, 0, frame16, SCREEN_WIDTH, S
CREEN_HEIGHT, WHITE);
            cute.play(S_SAD);
        }
        display.display();
        delay(FRAME_DURATION);
        currentFrame = (currentFrame + 1) % NUM_FRAMES_CRYIN
G;
```

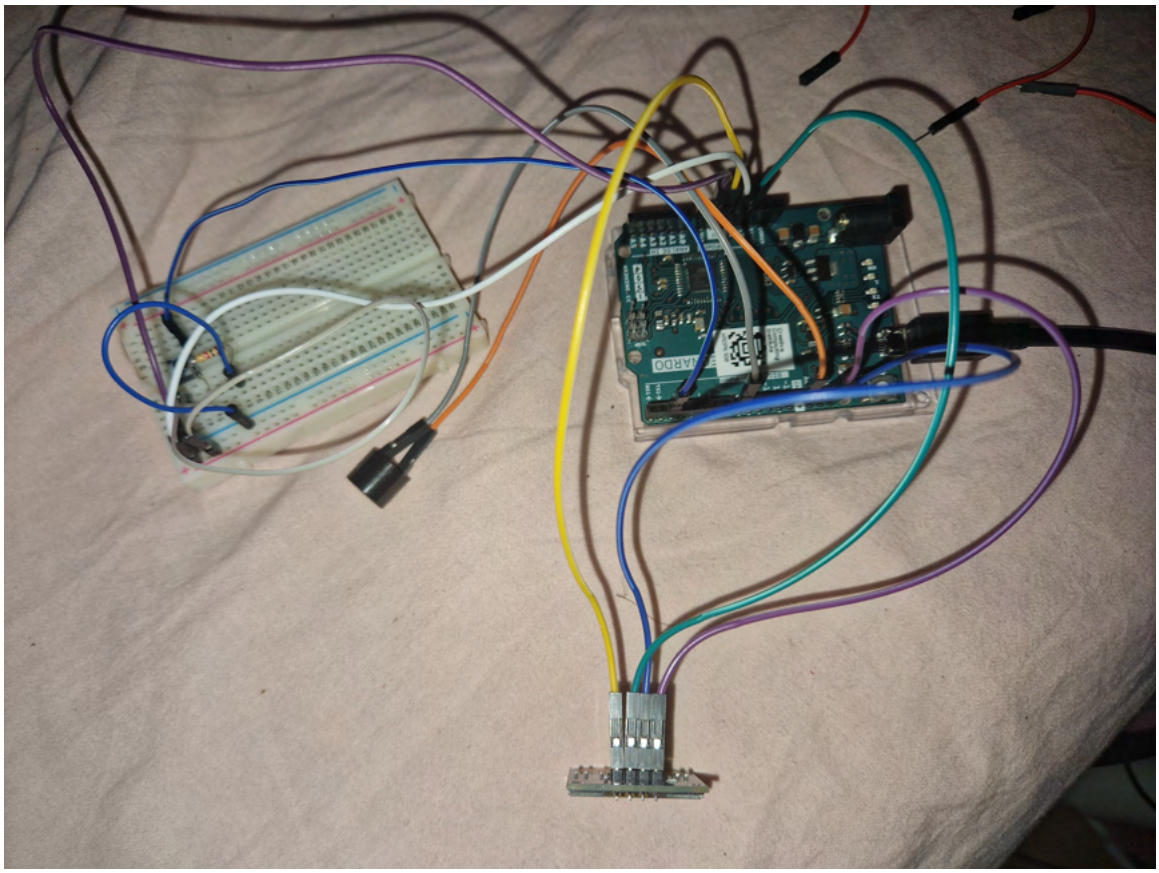
```
    }

    if (isMad) {
        // Mad animation
        display.clearDisplay();
        if (currentFrame == 0) {
            display.drawBitmap(0, 0, frame17, SCREEN_WIDTH, S
CREEN_HEIGHT, WHITE);
        } else if (currentFrame == 1) {
            display.drawBitmap(0, 0, frame18, SCREEN_WIDTH, S
CREEN_HEIGHT, WHITE);
        } else if (currentFrame == 2) {
            display.drawBitmap(0, 0, frame19, SCREEN_WIDTH, S
CREEN_HEIGHT, WHITE);
        } else if (currentFrame == 3) {
            display.drawBitmap(0, 0, frame20, SCREEN_WIDTH, S
CREEN_HEIGHT, WHITE);
            cute.play(S_CONFUSED);
        }
        display.display();
        delay(FRAME_DURATION);
        currentFrame = (currentFrame + 1) % NUM_FRAMES_MAD;
    }
}
```

The final code which is included on my git hub repository includes the sleep functions :

https://github.com/Brainrotz/CreativemakingArduinoCode/tree/main

final circuit (tinkercad did not have the correct oled screen for me to make it properly on it)



Thing I struggled with:

I severely underestimated the storage capacity of our provided arduino boards and had to cut back a lot of content I wanted to include.

I ran into a few problems I couldn't solve like the scaling issue, due to this I had to downsize the animation to 128 × 32 which I think affects the quality and resolution of the animations , if I was to redo this project I would reach out for help and spend more time (than I already did) editing and experimenting with the scaling.

Another big issue I ran into was getting the sleep animation to run properly so that when the button is pressed, it interrupts the loop . In the end this worked some what but now as smoothly as I wanted it to, I had originally wanted it to only be available when not in the idle animation is running but was unable to figure out how to do that.

Overall this was a challenging project in ways I completely did not expect but even with the errors I'm happy with the out come

If I was to redo this project I would spend time optimising the way the images are displayed and how the loops run to save RAM and have better execution of the code.