IM3180 Design and Innovation Project
(AY2023/24 Semester 1)
Group 3 Project Report

Title: **nIEMtendo**
Github: **https://github.com/angellshx/IEMDIP2023S1-Interactive-nIEMtendo.git**
Submitted by: **Samuel Loon**
Supervisor: **Professor Cheng Tee Hiang**

Abstract

nIEMtendo features two distinct games: the classic Snake game (Snake A) and our innovative version, Snake B. Offering three unique ways to play—through buttons, gesture sensors, and computer vision—our platform caters to diverse gaming preferences.

The classic Snake game (Snake A), a timeless arcade favourite, involves players guiding a growing snake to consume scattered food items, steadily increasing its length. Navigating the confined space without collisions becomes progressively challenging, demanding quick reflexes and strategic manoeuvres.

Snake B, our adaptation of the classic, introduces a novel twist. Instead of accelerating the game as the snake grows, we maintain its size. The challenge lies in identifying and consuming apples of varying colours scattered across the screen. Eating the correct-coloured apple is crucial to prevent the snake's demise, adding a layer of complexity and strategy to the gameplay.

Table of Contents

1. Background and Motivation

Reflecting on our fond memories of the classic mobile game Snake from our childhood, we were driven by the desire to introduce a contemporary twist, aiming to deliver a more engaging and interactive gaming experience for players. Our inspiration sprouted from modern gaming innovations, particularly the likes of the Wii, which seamlessly integrates game consoles with sensors to broaden the spectrum of game types available. It became clear to us that fusing our beloved retro games with cutting-edge technology could offer the newer generation a taste of the childhood experiences we cherished.

The inception of our idea involved merging the simplicity of the snake game, a nostalgic favourite, with advanced gesture sensors and computer vision technologies. By infusing these elements, we envisioned creating not only the classic snake game but also a colourful, unique mode developed by our team. Our goal is to transport players back to the joyous moments of their childhood while embracing the possibilities of contemporary gaming.

Our project is not just about reviving the past; it's a forward-looking endeavour to expand the realm of gesture recognition in arcade games. Through our initiative, we aim to introduce a diverse array of games that can be enjoyed in arcade settings, bringing a refreshing twist to traditional gaming experiences.

1.1 Project Delegation

Team	Main Job Scope	Members
Project Leader	Overseeing the collaboration among the hardware, software (game and design), and computer vision subgroups	- Samuel Loon
Hardware	Wiring, soldering, circuit analysis, mounting of the board and cutting of wood	- Ong Yu Heng - Eliza Tanpoco - Ang Kai Xun
Software (Game)	Writing codes of the games and display of the small screens, did testing and debugging to ensure error-free experience	- Heidi Lee - Angel Low
Software (Design)	Writing codes of Main Menu Screen and coming up with the design for the exterior of the board	- Claire Chai - Joshua Pok
Computer Vision	Writing codes with OpenCV for gesture detection	- Wen Xi Xiong - Ng Zheng NIng

2. Objective

Our project aims to craft a distinctive interactive experience that resonates with audiences of all ages. By revitalising the timeless snake game from our collective past, we incorporate gesture sensors and computer vision technology, enabling individuals to nostalgically relive the simple joys of childhood. Our aspiration is to bring people of diverse age groups together through this arcade game, fostering the creation of cherished memories for everyone involved.

3. Review of Literature/Technology

Before the start of the project, we first researched on the features we would like to include. Research was done to find out more on Gesture Sensors and Computer Vision.

3.1 Hardware Preparation

The successful execution of the Snake Game project heavily relies on the appropriate selection and setup of hardware components. This section delves into the key aspects of hardware preparation that form the backbone of the project.

3.1.1 LEDs

The LED WS2812B constitutes a fundamental hardware component in the project's LED matrix setup. Renowned for its individually addressable RGB LEDs, the WS2812B offers versatility in illuminating each diode independently. These LEDs are designed for series connection, allowing a daisy-chaining mechanism where each LED's data output connects to the input of the subsequent LED. This configuration enables seamless communication with a single data line from Arduino simplifying the wiring architecture and reducing the required GPIO pins.

3.1.2 Hardware Preparation

We measured and cut the LED strips and pasted it on the mounting board. As we wanted to have better pixels for the game, we have decided to go with a sizing of 28 by 20. This enhances the overall visual experience for better gameplay.

Soldering was done to connect the strips and wires were colour coded for easy reference. Red was used for 5V, Blue for Data transmission and Black for ground. The wiring of the start of the LED strip was soldered to longer wires and coiled up. This is to ensure a clean and organized wiring layout to accommodate the length of the board.

To achieve proper light diffusion, a grid setup is implemented, ensuring each LED is precisely positioned within the designated grid. Acrylic and frosted matte materials were used as effective light diffusion.

3.2 Gesture Sensors

We implement the PAJ7620U2 gesture sensor module to our project. This module can recognize 9 gestures including move up, move down, move left, move right, etc with a simple swipe of your hand. Compared with solutions like APDS-9960, this module is faster, more accurate, while recognising more gestures with higher anti-interference capability. It suits low power applications such as smart home, robot interaction, etc.

3.3 Open Computer Vision

OpenCV (Open Source Computer Vision Library) is an open-source computer vision and machine learning software library. OpenCV was built to provide a common infrastructure for computer vision applications and to accelerate the use of machine perception in commercial products. The library has more than 2500 optimised algorithms, which includes a comprehensive set of both classic and state-of-the-art computer vision and machine learning algorithms.

These algorithms can be used to detect and recognise faces, identify objects, classify human actions in videos, track camera movements, track moving objects, extract 3D models of objects, produce 3D point clouds from stereo cameras, stitch images together to produce a high-resolution image of an entire scene, find similar images from an image database, remove red eyes from images taken using flash, follow eye movements, recognise scenery and establish markers to overlay it with augmented reality, etc.

In our project, openCV was used to capture the frames from the Pi camera. After that, MediaPipe's hand-tracking model was used to detect the direction of the hand. MediaPipe's hand-tracking model will return the coordinates of hand landmarks. We then can use the coordinates to determine the direction of the hand.

4. Design and Implementation

4.1 Design Consideration / Choice of components

In this project, nIEMtendo aims to act as a user-interactive game platform, which reflects a classic experience at an arcade. More information can be obtained in the table as follows:

User Experience	The design prioritises delivering a nostalgic experience for users who remember the classic Snake game. User interface elements, inclusive of the game mechanics and its visual aesthetics should evoke fond memories while ensuring an enjoyable and engaging experience.
Integration of Gesture Sensors	Gesture sensors introduce a hands-free and intuitive control mechanism, where players can navigate the snake by making specific hand motions,

	adding an element of physical engagement to the gameplay. Players should expect immediate and accurate recognition of their motion to control the snake effectively.
Integration of Buttons	Buttons provide players with a versatile and tactile input. Its responsiveness is essential for quick and precise actions. The tactile feedback from pressing the buttons contributes to a more immersive experience.
Integration of Computer Vision	Computer vision technology allows for a more dynamic and immersive gaming environment. Hand recognition can be used to introduce an immersive gaming experience.
Accessibility and Inclusivity	The different multiple control options allow players to choose either a hands-free experience or traditional button-based controls, which ensures inclusivity in both options. There are audio cues and visual feedback to inform players of game stages, on top of visual indicators that show the game state.
Educational and Entertaining	The blending of Arduino (programming) and integration with hardware uses technology to create games that are both fun and intellectually stimulating.
Market Trends and Audience Appeal	Gaming continues to dominate the market, where there is a growing demand for innovative and unique gameplay interactions. nIEMtendo produces a fresh and interactive twist with the introduction of controls and computer vision to a classic game, aligning with the trend of pushing boundaries in gameplay mechanics. Furthermore, there is a continued interest in nostalgia, with many gamers still enjoying retro gaming experiences, which nIEMtendo taps into the sentimental value while combining with modern technology.
Cost and Resource Management	The cost implicated in the building of the product is within the allocated budget and resources of IEM Workshop @ NTU and $500.

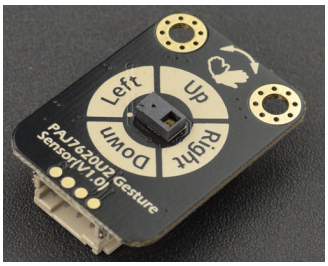
In addition to the mentioned aspects, we have consulted and drawn inspiration from previous cohorts of students who have undertaken this module:

	The product labelled “GAMEBOY MACRO” has successfully integrated buttons and the screen in the middle. However, the following were ideas that we had discussed and improved on: 1. The touch sensors (on the left) were placed in a position that was not allowing the single user to press the buttons on the other end of the product (on the right). 2. The alignment of the WS28128B LED (or otherwise) that they used were not aligned appropriately and was very visible despite the acrylic on the top.
	The “SLOT MACHINE” successfully integrated a user-friendly button-in-front-of-the-screen placement. Hence we took inspiration from it and implemented it into our product. However, they used an LCD screen, which was expensive. Hence, we did not want to pursue the same to spend beyond our budget.

	The “TETRIS” product successfully integrated the tetris game, with a clear score, and high score screens. Nevertheless, the demonstration encounters delays during gameplay, diminishing the overall positive user experience. Furthermore, while the frosted acrylic positioned over the screen facilitates light diffusion, there are clear misalignment issues, and its large cubes resulted in a low-quality diffusion effect. “TETRIS”, in fact, does not necessitate the use of up and down buttons, given the presence of dedicated buttons for both dropping and altering the shape of tiles. Thus, from this we did not want to specify the name of our game, as we wanted the console to be able to cater to more games, in future if possible, to more than just the Snake game.

The choice of components for nIEMtendo, derived from our own ideation and inspiration from other products, we have concluded with the following:

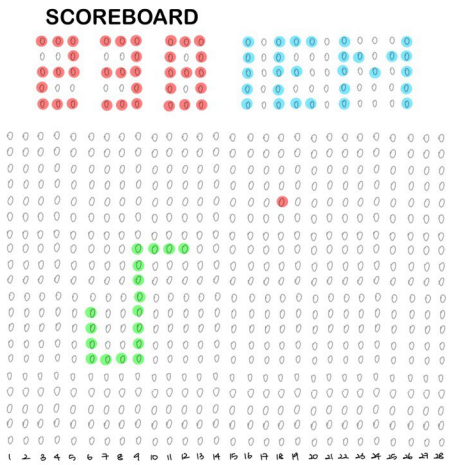
WS2818B LED Strips 	High-quality programmable LED strip lights compatible with Arduino and various other microcontrollers.
Mounting Board 	Utilized to organize WS2818B LED Strips in accurately measured rows.

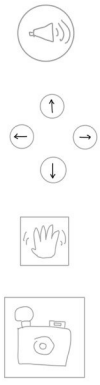
Gesture Sensor PAJ7620U 	A gesture recognition module with a universal I2C interface, consolidated into a single chip capable of recognising 9 gestures. It incorporates an embedded infrared LED and optical lens, ensuring functionality, even in low-light and dark environments.
Raspberry Pi Camera Module 	Enables high-definition video recording when connected to a Raspberry Pi.
Plywood 	Employed in constructing the product's casing, providing a sturdy framework.
Speaker Module 	Facilitates the connection of speakers to Arduino microcontrollers.
Arduino Mega 2560 CH340 	An Arduino-based microcontroller on the ATmega2560, featuring 54 digital input/output pins. With expanded memory and increased digital/analog pins, the Mega facilitates handling complex game logic, enabling smoother processing. Its extended capacity allows for advanced functionalities, more intricate algorithms, and enhanced data storage. The Mega's surplus GPIO pins offer versatility in interfacing with diverse components critical for gesture recognition, LED matrix control, and other vital features.

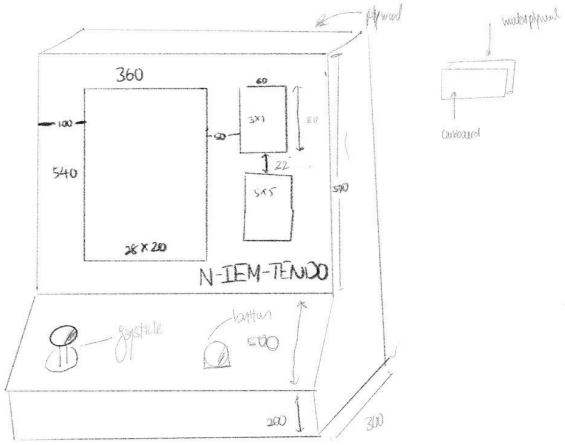
Raspberry Pi 	A cost-effective computer designed to plug into a monitor, used for processing Computer Vision in relation to our product.
Acrylic Sheet 	Employed as the material for our screen.

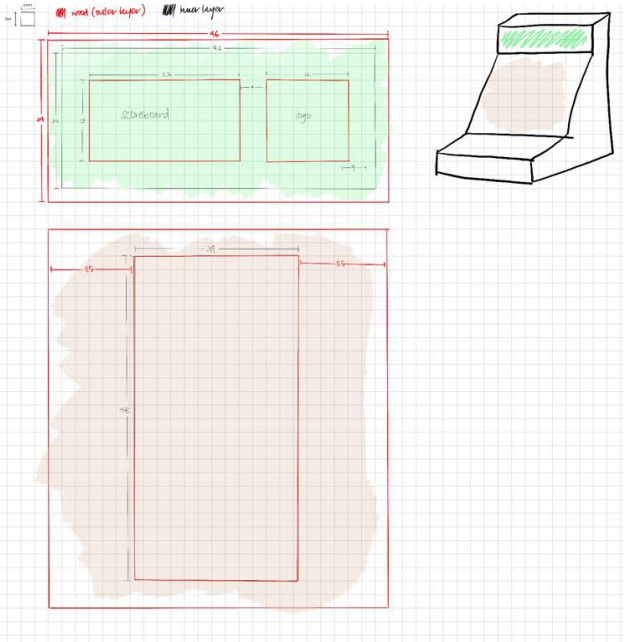
4.2 Final Design

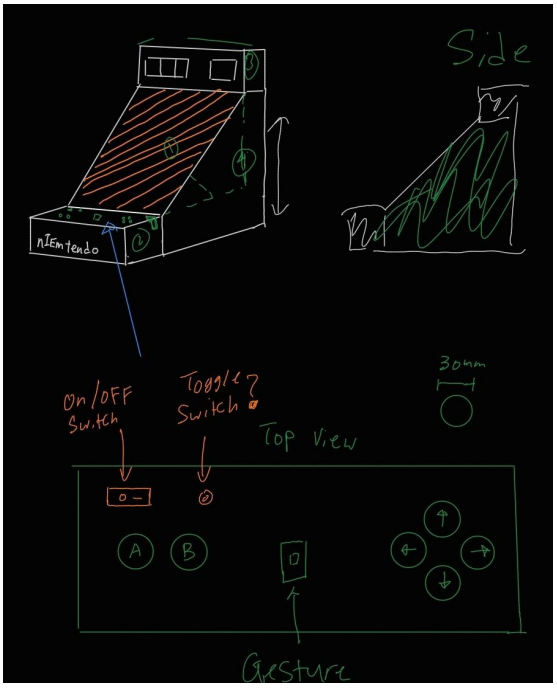
4.2.1 Early sketches and ideas











4.2.2 Finished product

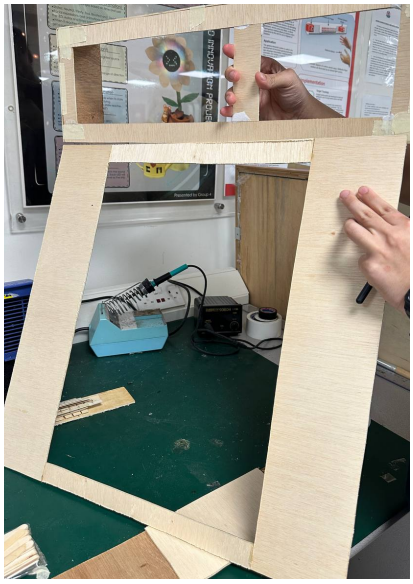
4.3 Implementation

4.3.1 Hardware

Our gaming machine's construction was a meticulous process, divided into distinct phases focusing on the body, the grid, and the LED panel. Each component was carefully designed and assembled to ensure both functionality and aesthetic appeal.

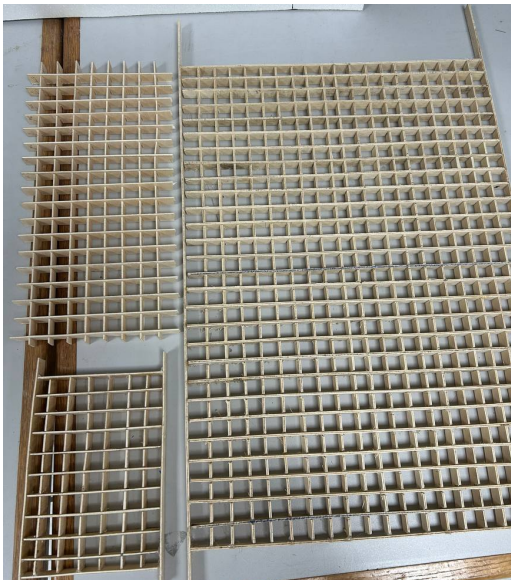
4.3.1.1 Body Construction

The body of the machine was crafted using 0.5 cm thick wood, repurposed from leftovers from a previous project group. This not only demonstrated our commitment to sustainability but also provided a sturdy and reliable material for our machine's frame. The wood pieces were precisely cut using the band saw in our workshop, ensuring accurate dimensions and clean edges. The cut pieces were then assembled using wood glue, forming a robust structure. Post-assembly, the body was spray-painted in black, giving it a sleek and professional finish. This choice of colour not only adds to the machine's visual appeal but also enhances its ability to blend with various environments.



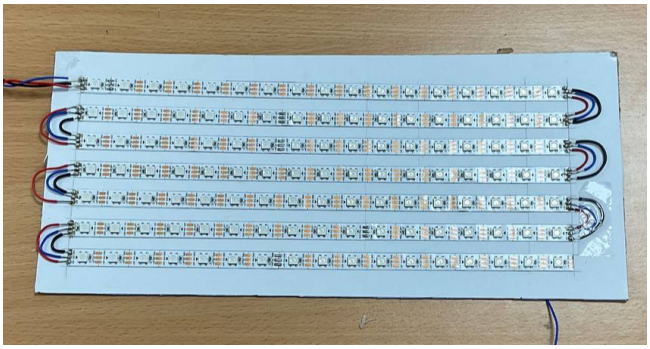
4.3.1.2 Grid Design

For the grid, we opted for 1.5mm plywood, a material known for its light weight and durability. The plywood was cut into 1.5 cm long strips, which were then interlocked over the LED panel. This grid structure was instrumental in creating individual compartments for each LED bulb, facilitating a controlled and uniform spread of light. The cubic light diffusion achieved through this design was further refined by placing a sheet of translucent acrylic over the grid. This additional layer played a crucial role in diffusing the light, softening the LED panel's resolution, and contributing to a more visually appealing display.

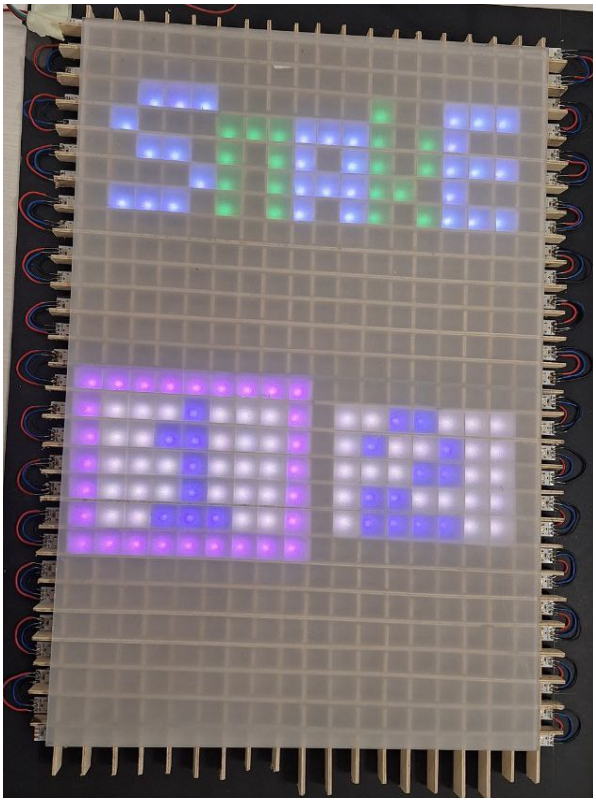


4.3.1.3 LED Panel Assembly

The LED panel is a critical component, serving as the heart of the machine's visual display. Strips of LEDs were carefully soldered onto a piece of cardboard, with the rows alternating to ensure an even distribution of light. The entire assembly was then connected to and controlled by a central Arduino unit. This choice of control system allowed for a high degree of customization and flexibility in programming the LED behaviour, making it possible to create a wide range of visual effects and patterns. The Arduino's central control ensures that the LEDs function in a coordinated and efficient manner, enhancing the overall user experience.



The implementation of our gaming machine was a fusion of innovative design and practical engineering. Each component – the body, the grid, and the LED panel – was crafted with attention to detail and a focus on quality. By repurposing materials, employing precise fabrication techniques, and integrating sophisticated electronics, we have succeeded in creating a gaming machine that is not only visually striking but also robust and versatile in its functionality.



4.3.2 Software

4.3.2.1 Process Timeline

The software team divided responsibilities, with one subgroup focusing on researching and coding for the screens, and the other tackling the game development. Given the team's unfamiliarity with Arduino, a two-week period was dedicated to intensive research and learning, encompassing both coding with Arduino and understanding the specific components assigned to each subgroup. This concerted effort not only facilitated skill development but also ensured a comprehensive understanding of the hardware and software components, laying a solid foundation for the subsequent phases of the project.

Item	1	2	3	4	5	6	7	8	9	10	11	12	13
Research & Learning Arduino													
Main Menu													
Mini LED Screens (Score board etc.)													

Snake Game A													
Snake Game B													
Sound Effects													
Finishing Touches, Testing													

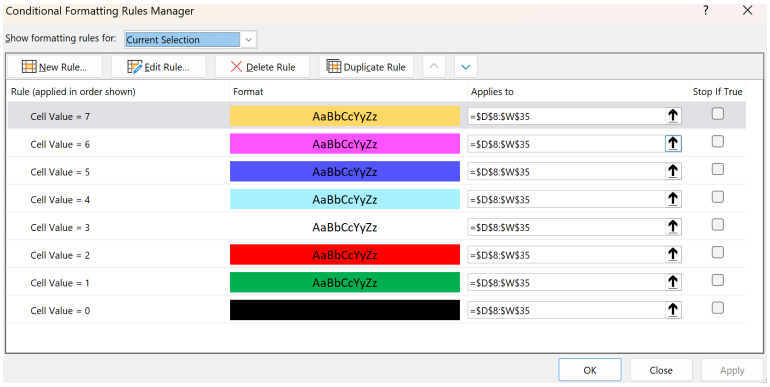
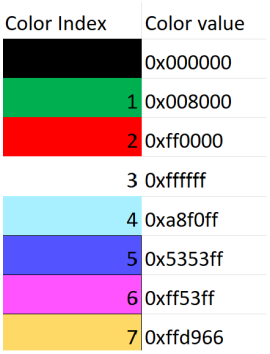
4.3.2.2 Required libraries

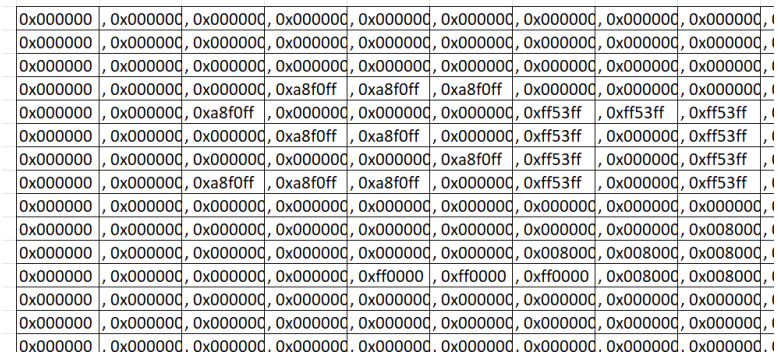
	Libraries Used	**Descriptions**
Visual	FastLED.h	● Used for main menu and game display ● Fast, efficient, compatible LED driver code for wide range of addressable LEDs and arduino boards ● Provides full HSV colour support as well as classic RGB ● Master brightness setting (nondestructive) controls brightness, power use, and battery life
	Adafruit.h	● Used for small displays (score/colour display, highscore/timer display)
Gameplay	LinkedList.h	● Store X, Y values to point starting points of snake and apples ● Create snake pattern: used in defining snake length using pointer system
	DFRobot_PAJ7620U2.h	● Control Gesture Sensor to change snake's direction
	EEPROM.h	● Storing values in memory for highscores of snake games A and B
Audio	SoftwareSerial.h	● Allows serial communication on other digital pins of an Arduino board

Table 1: Arduino Libraries Used

4.3.2.3 Creating Start menu design

To design the menu screen, we made use of Microsoft Excel to help us visualise and map out the individual pixels.

Tools used	Usage
 	In order to automate the process, we employed conditional formatting to assign a unique colour output based on an entered number (eg. 1, 2, 3, etc), eliminating the need for manual colour selection. Thereafter, we assign each number to a hex colour code and plot it out on the excel sheet.
	Excel sheet to visualise and design the menu screen.

	Using a separate sheet and a formula, we were then able to easily transfer the hex colour codes into the Arduino IDE to render the colours out on the screen.

4.3.2.4 Integration of different boards and game mode codes:

Split into 5 different arduino files for easier debugging and better organisation, the usage of each sub file is as follows:

Files	**Usage**
snake.ino	• Declare global variables for main gameplay • Include all libraries (except Adafruit) • Initialise main setup() • Includes gameLoop() which contains code functionalities for snake v1 and v2 Snake Game A: • 1 Linked list for Snake Coordinates • 1 Apple, no Linked List • 1 LED Colour for Apple Snake Game B: • 1 Linked list for Snake Coordinates • 4 Linked Lists for Apple Coordinates • 5 LED Colours for 10 Apples (number of Apples can be changed)
menuDisplay.ino	• Contains menuSetup() which initialises buttons A and B to enter snake games • Contains colour matrix for start menu

	• Initialises main loop() which contains frame switching logic to enter different game modes according to button pressed
scoredisplay.ino	• Include Adafruit library • scoreSystem() setups score/colour board display in setup() • displayScore() function called from playGame() if in snake A • displayColor() function called from playGame() if in snake B
timerDisplay.ino	• timerdisplay() setups timer/highscore board display in setup() • displayHighScore(1) function called from gameLoop() if in snake A • displayHighScore(2) function called from gameLoop() if in snake B • displaytimer() function called from gameLoop() if in snake B
music.ino	• Include SoftwareSerial library • Code functionalities to convert mp3 music files from SD card to be recognised by arduino

4.3.3 CV

4.3.3.1 Installation process

Python: Mediapip-rpi4(For raspberry Pi 3) Mediapip-rpi4(For raspberry Pi 4) Gtts Mpg321 Numpy Serial Picamera2 OpenCV	These libraries can be installed by using the “pip” command.

Compiling OpenCV on Raspberry Pi:
$ free -m
$ wget https://github.com/Qengineering/Install-OpenCV-Raspberry-Pi-64-bits/raw/main/OpenCV-4-5-0.sh
$ sudo chmod 755 ./OpenCV-4-5-0.sh
$ ./OpenCV-4-5-0.sh
	The compiling process may vary due to different OS versions and changes in dependencies' names. If any error while executing the compiling script, changing the dependency's name to the latest will solve the issue.
	
	The compilation process can take up to 1 and a half hours for Raspberry Pi 4 and 3 to 4 hours for Raspberry Pi 3.

Arduino IDE for Linux arm 64	Can be downloaded from the website of Arduino.

4.3.3.2 Code Implementation

Raspberry Pi	
cvvv2.py	This is the main Python file which tracks the hand motions and sends the direction to Arduino.
track_hands()	This function tracks the coordinates of hand landmarks and processes them into directions.
setupSerial(baudRate, serialPortName)	This function sets up the Serial Communication with the Arduino. Make sure the baudRate and serialPortName are the same as shown in Arduino IDE.
sendToArduino(stringToSend)	This function sends the message with startMarker and endMarker to Arduino so that Arduino can process the actual message without concatenating or missing messages.
waitForArduino()	This function simply waits until the Arduino sends 'Arduino is ready'. It also ensures that any bytes left over from a previous message are discarded
recvLikeArduino()	This function reads the reply from Arduino, it is more for debugging purposes.
Arduino	
Snake.ino	This is the Arduino code, we will only discuss the part of how it interacts with Python in this section.
replyToPython()	This function sends the reply message to Raspberry Pi. It also acknowledges the successful delivery of the previous message. It also assigns the received message to the variable "str", which is the input of the direction.
recvWithStartEndMarkers()	This function will receive the message sent from Raspberry Pi and extract the content from the start and end markers.

5. Conclusion and Recommendation

5.1 Conclusion

In conclusion, the nIEMtendo project represents a harmonious blend of nostalgia and innovation, as it delivers a captivating gaming experience that transcends traditional boundaries, while including modern technologies. By reviving the classic Snake game and infusing it with modern technologies, we have created a platform that not only pays homage to arcade favourites, but also pushes the limitations of interactive gameplay.

Our commitment to user experience shows through our incorporation of diverse control mechanisms, catering to both traditionalists who favour buttons and enthusiasts who embrace the hands-free interaction provided by gesture sensors and computer vision. This results in a gaming environment that is not only immersive but also accessible to a wide audience, fostering inclusivity in the world of gaming.

Technological advancements, as we integrated WS2818B LED strips, PAJ7620U gesture sensor, Raspberry Pi camera module, and Arduino Mega 2560 CH340, have elevated nIEMtendo to new heights. This project stands as a testament to our team's strong capabilities in leveraging cutting-edge hardware to create a visually stunning and intellectually stimulating gaming experience.

On top of this, we have also been successfully cost-effectively managing resources within the allocated budget. This highlights our commitment to efficiency without compromising quality. This accomplishment positions nIEMtendo as a viable and scalable solution that opens doors for future projects within similar constraints.

Looking forward, the recommendations for future works provide a roadmap for continued innovation, from expanding game content and enhancing computer vision features to exploring community engagement. nIEMtendo is not merely a project; it is a catalyst for the future of interactive gaming, education and technological exploration.

Lastly, we extend our greatest gratitude to the IEM Workshop @ NTU, team members, advisors (Professor Cheng TH and Lee-Tay Annie), Garage @ EEE, and Robotics Workshop @ NTU, who have played pivotal roles in bringing nIEMtendo to fruition.

As we conclude this chapter, we eagerly anticipate the journey ahead, confident that nIEMtendo has the potential to leave an indelible mark on the Information Engineering and Media landscape.

Thank you for reading, and joining us, on this exciting journey of nIEMtendo.

5.2 Recommendation for Future Works

In view of our project, here are the recommendations for future works:

User feedback and testing	Conduct extensive user testing to gather feedback on the user experiences to identify any usability issues, preferences, or additional features that users may desire.
Enhancements based on feedbacks	Implement improvements and modifications based on user feedback, involving refining control mechanisms, adjusting difficulty levels, or adding new features to enhance overall gameplay.
Multiplayer features	Explore the possibility of adding multiplayer features to the games, which could include competitive or cooperative modes, allowing users to play with friends or other players online.
Expand game content	Introduce additional levels, challenges, or game modes to keep players engaged.
Enhanced computer vision features	Explore advanced computer vision features to further enhance the gaming experiences, such as by including more sophisticated hand gestures or recognition, for personalised user interactions.
Mobile application integration	Develop a mobile application that complements the gaming experience, which could serve as a controller for the game, provide additional information, or even offer mini-games related to nIEMtendo.
Community engagement	Build a community around nIEMtendo by creating social media channels where players can share their experiences, tips and suggestions. Also, consider organising events or competitions related to nIEMtendo to keep the community engaged.

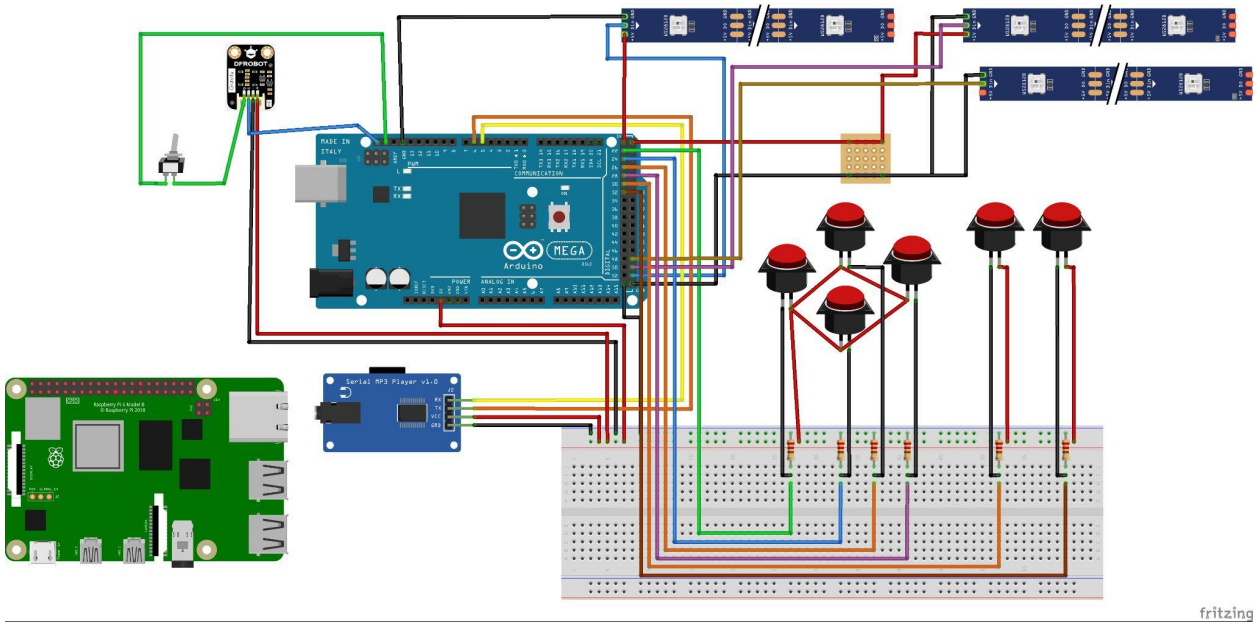
Appendices:

1. Bill of Materials (BOM)

S/N	Item	Purpose	Quantity	Price	Remarks
1	WS28128FB LED Strip	-	1	$40	Isn't compatible with arduino
2	WS28128 LED Strip (Retail)	To test LED strip	1 x 2m	$24	-
3	WS28128 LED Strip (Online)	Display screen	3 x 5m	$35.30 (incl. shipping)	-
4	Mounting Board	To place LED	2	$6	-
5	Gesture Sensor (PAJ7620U2)	Using hand to control snake movement	1	$26.20	-
6	Camera module OV7670	-	1	$6.15 (incl. shipping)	Didn't use
7	Plywood (1.5mm)	To create the grid	12	$32.72	-
8	Speaker Module	Connection to Speakers and Arduino	1	$4.22 (incl. shipping)	-
9	Arduino Mega 2560 CH340	Processing and controlling electronic components.	1	$16.95 (incl. shipping)	-
10	Raspberry Pi 4 Model (8GB)	Computation and data processing	1	$119.99	-
11	Raspberry Pi Camera Module	Detect overall hand movement to control the snake	1	$44.64	-
12	Raspberry USB Plug	To power up the Raspberry Pi	1	$14.00	-
13	Raspberry Pi 15-pin Camera FFC Cable	To connect the Raspberry Pi with the Raspberry Pi Camera module	1	$3.89	-
14	Buttons	Move the snake and select different game	10	$5.75 (incl. shipping)	-

		modes			
15	Plywood (3mm)	Outer Structure	3	$15	-
16	HDMI to Mini HDMI Cable	Connect Raspberry PI module to the game monitor	1	$4.82 (incl. shipping)	
17	MicroSD for Raspberry PI	Storing data	1	$9.74	
25	FASTENERS (Bracket,Hinge,Screw,Nut**)**	Providing structural integrity	18	$24.4	
26	Acrylic Sheet	Use for diffusion of light	2	25.34 (incl. shipping)	
27	Matte Sticker	Use for diffusion of light	1	$6.34 (incl. shipping)	
28	Speakers	Sound output	2	Lab	
29	Styrofoam	Base of product	1	Lab	-
30	Toggle switch	To activate gesture sensor	1	Lab	
31	Paint	Paint out the snake design	-	Lab	
32	PaintBrush	Use to paint	2	Lab	
33	Masking Tape	Labelling	1	Lab	
34	Spray Paint	Paint the outer wood	1	$3.90	
Total				**$469.35**	

2. Design Diagrams



Module	Pin	Module	Pin
Main Screen LED	53	Button Down	22
Score Screen LED	51	Button Right	24
Secondary Screen LED	49	Button Left	26
Gesture Sensor	SDA,SCL	Button Up	28
Speaker Module	RX:5, TX:6	Button A	30
Raspberry Pi 4	USB Cable	Button B	32

3. User Guide

1. Player will start at menu screen
    a. Select A button for Snake A , select B button for Snake B.
    b. To activate Gesture sensor, toggle switch down
    c. To activate OpenCV, open the camera hole

2. Snake Game A gameplay:
    a. The player uses the arrow buttons/hand gestures to move a "snake" around the board.

b. As the snake eats the apple, it grows larger and its speed increases as it grows as well.
c. The game ends when the snake either moves off the screen(aka collides into the walls of screen) or moves into itself.
d. The goal is to grow the snake as large as possible before that happens.
e. Additional screen displays:
    i. Scoreboard that increments by 1 every time it successfully eats an apple and grows
    ii. Highscore by previous players

3. Snake Game B gameplay:
    a. The player uses the arrow buttons/hand gestures to move a "snake" around the board.
    b. There will be an assigned colour apple that the player is supposed to eat.
    c. The game ends when the snake either moves off the screen(aka collides into the walls of screen) or moves into itself or eats the wrong colour apple.
    d. The goal is to finish eating all the apples.
    e. Additional screen displays:
        i. Color board that indicates what colour apple to eat
        ii. Countup timer

4. Both games will return to starting menu screen upon losing or winning
    a. Player can choose to replay the same game or select a different snake game mode to play

4. Maintenance Guide

4.1. Power Management

- Ensure the machine is connected to a stable power source.
- Periodically check power cables for wear and tear.
- Implement a power surge protector for additional protection.

4.2. Cleaning and Dust Prevention

- Regularly clean the cabinet, control panel, and other external components.
- Keep the machine in a clean and well-ventilated environment.

4.3. Operating System Updates

- Keep the operating system of the Raspberry Pi up to date.
- Ensure compatibility with the latest firmware for Arduino and Raspberry Pi.

4.4. Audio System Check

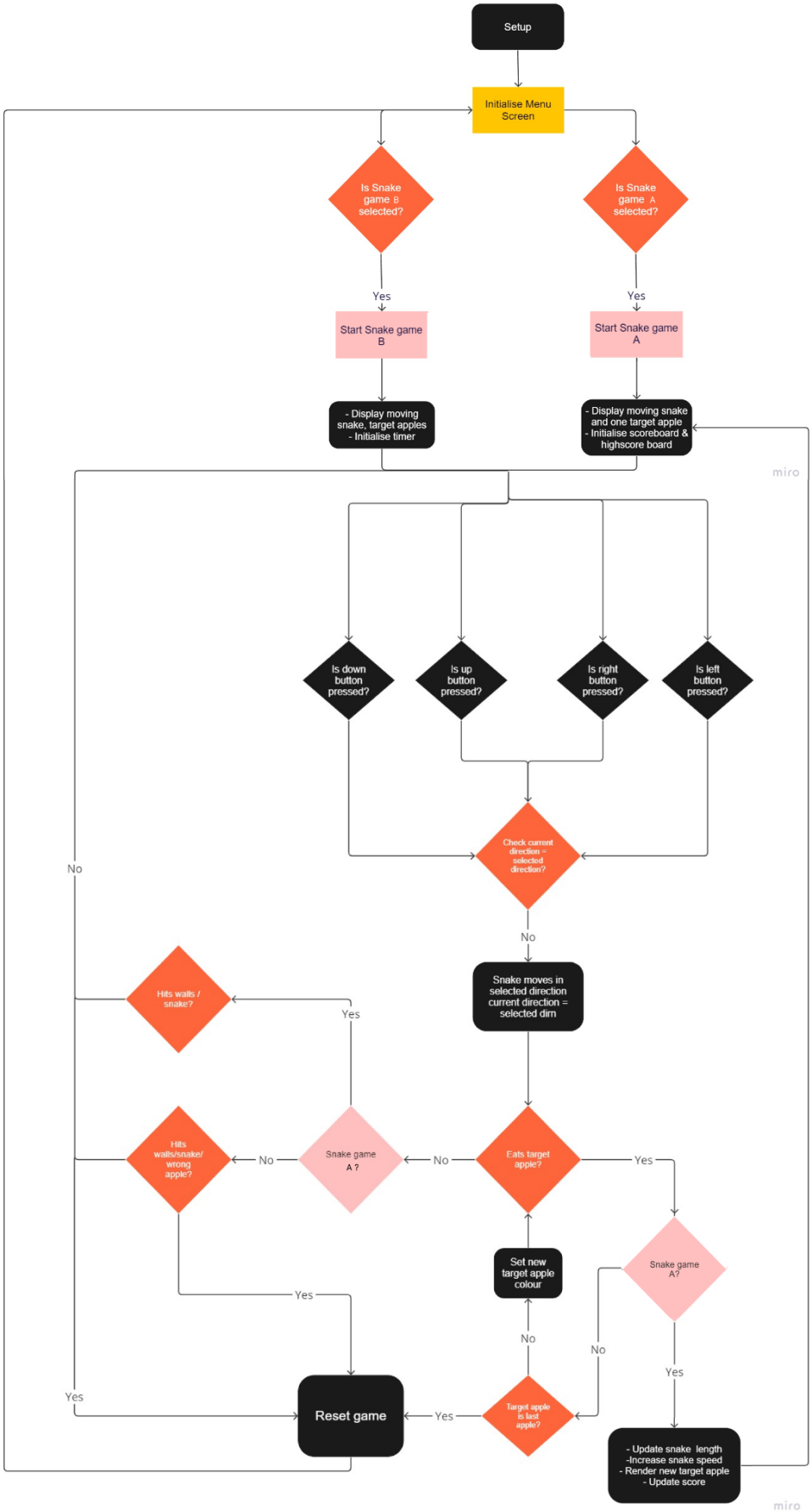
- Test and adjust audio levels.
- Verify the functionality of speakers and audio connections.

4.5. System Crashes

- Open the machine and turn off both Raspberry Pi and Arduino.
- Restart the machine and see if the problem resolves.
- If not, please reformat the Raspberry and download all the necessary libraries and programs in our GitHub Repository.

5. How-To Guides

Flowchart



6. Source Code

Computer Vision Source Code

```
import cv2
import mediapipe as mp
import time
import threading
import serial
from picamera2 import Picamera2

mp_drawing = mp.solutions.drawing_utils
mp_hands = mp.solutions.hands

picam2 = Picamera2()
picam2.configure(picam2.create_preview_configuration(main={"format":"RGB
888","size":(640,480)}))
picam2.start()
width = 640
height = 480

#ser = serial.Serial('/dev/ttyACM0',115200)
#ser.reset_input_buffer()

cv2.startWindowThread()

# Initialize hand tracking with increased min_detection_confidence
hands = mp_hands.Hands(
    static_image_mode=False,
    max_num_hands=1,  # Detect only one hand
    min_detection_confidence=0.4,  # Increase the confidence threshold
    min_tracking_confidence=0.5,
)

startMarker = '<'
endMarker = '>'
dataStarted = False
dataBuf = ""
messageComplete = False

#=======================
#=======================
```

```
    # the functions

def setupSerial(baudRate, serialPortName):

    global  serialPort

    serialPort = serial.Serial(port= serialPortName, baudrate =
baudRate, timeout=0, rtscts=True)

    print("Serial port " + serialPortName + " opened  Baudrate " +
str(baudRate))

    waitForArduino()

#========================

def sendToArduino(stringToSend):

        # this adds the start- and end-markers before sending
    global startMarker, endMarker, serialPort

    stringWithMarkers = (startMarker)
    stringWithMarkers += stringToSend
    stringWithMarkers += (endMarker)

    serialPort.write(stringWithMarkers.encode('utf-8')) # encode needed
for Python3


#==================

def recvLikeArduino():

    global startMarker, endMarker, serialPort, dataStarted, dataBuf,
messageComplete

    if serialPort.inWaiting() > 0 and messageComplete == False:
        x = serialPort.read().decode("utf-8") # decode needed for
Python3

        if dataStarted == True:
            if x != endMarker:
```

```
                dataBuf = dataBuf + x
            else:
                dataStarted = False
                messageComplete = True
        elif x == startMarker:
            dataBuf = ''
            dataStarted = True

    if (messageComplete == True):
        messageComplete = False
        return dataBuf
    else:
        return "XXX"

#==================

def waitForArduino():

    # wait until the Arduino sends 'Arduino is ready' - allows time for
Arduino reset
    # it also ensures that any bytes left over from a previous message
are discarded

    print("Waiting for Arduino to reset")

    msg = ""
    while msg.find("Arduino is ready") == -1:
        msg = recvLikeArduino()
        if not (msg == 'XXX'):
            print(msg)


def track_hands():
#     cap = cv2.VideoCapture(0)

    history_length = 5  # Length of position history
    threshold_x = 50  # Decreased threshold for horizontal movement
    threshold_y = 5  # Decreased threshold for vertical movement
    hand_x_position_history = [0] * history_length
    hand_y_position_history = [0] * history_length
```

```
    current_direction = ""
    setupSerial(115200, "/dev/ttyUSB0")
    count = 0
    prevTime = time.time()

    while True:
        im = picam2.capture_array()
        results = hands.process(cv2.cvtColor(im, cv2.COLOR_BGR2RGB))

        #line = ser.readline()
        #print("Read " + line.decode('utf-8', errors='ignore').strip()+
"from arduino")

        if results.multi_hand_landmarks:
            hand_landmarks = results.multi_hand_landmarks[0]  # Use the
first detected hand

            tip_x_position =
hand_landmarks.landmark[mp_hands.HandLandmark.MIDDLE_FINGER_TIP].x *
width
            mcp_x_position =
hand_landmarks.landmark[mp_hands.HandLandmark.MIDDLE_FINGER_MCP].x *
width
            tip_y_position =
hand_landmarks.landmark[mp_hands.HandLandmark.MIDDLE_FINGER_TIP].y *
height
            mcp_y_position =
hand_landmarks.landmark[mp_hands.HandLandmark.MIDDLE_FINGER_MCP].y *
height

            hand_x_position_history.append(tip_x_position -
mcp_x_position)
            hand_y_position_history.append(tip_y_position -
mcp_y_position)

            if len(hand_x_position_history) > history_length:
                hand_x_position_history.pop(0)
            if len(hand_y_position_history) > history_length:
                hand_y_position_history.pop(0)

            smoothed_x_position = sum(hand_x_position_history) /
```

```
history_length
            smoothed_y_position = sum(hand_y_position_history) /
history_length

#              arduinoReply = recvLikeArduino()
#              if (arduinoReply == 'Reset'):
#                  serialPort.flush()
#                  serialPort.reset_input_buffer()
#                  serialPort.reset_output_buffer()

#              if not (arduinoReply == 'XXX'):
#                  print("Time %s  Reply %s" %(time.time(),
arduinoReply))

              if smoothed_x_position < -threshold_x:
                  current_direction = "C"
                #  ser.write("A\n".encode('utf-8'))
              elif smoothed_x_position > threshold_x:
                  current_direction = "A"
                #  ser.write("D\n".encode('utf-8'))
              elif smoothed_y_position < -threshold_y:
                  current_direction = "B"
                 # ser.write("W\n".encode('utf-8'))
              elif smoothed_y_position > threshold_y:
                  current_direction = "D"
                 # ser.write("S\n".encode('utf-8'))

              if time.time() - prevTime > 0.1:
                  sendToArduino(current_direction)
                  prevTime = time.time()


              # Clear the screen and display the current direction
              print("\033[H\033[J")
              if current_direction:
                  print("Move " + current_direction)


#                  ser.write(str(current_direction+"\n").encode('utf-8'))


#              mp_drawing.draw_landmarks(frame, hand_landmarks,
```

```
mp_hands.HAND_CONNECTIONS)

        cv2.imshow('Frame', im)
        if cv2.waitKey(1) & 0xFF == 27:
            break


    cv2.destroyAllWindows()

if __name__ == "__main__":
    tracking_thread = threading.Thread(target=track_hands)
    tracking_thread.start()

    tracking_thread.join()
```

snake.ino

```
//LEDs
#include <FastLED.h>
#include <LinkedList.h>
#include <DFRobot_PAJ7620U2.h>
#include <EEPROM.h>
#define NUM_LEDS 560
#define LED_PIN 53

unsigned long previousMillis = 0;
unsigned long previousMillis_reset;
const unsigned long interval = 250;

//reading serial
String str;


int rows = 28;
int columns = 20;
CRGB leds[NUM_LEDS];
int mode1 = true;
boolean newData = false;
const byte numChars = 128;
char receivedChars[numChars];
int gesInt = 0;
```

```
// Init gesture sensor
DFRobot_PAJ7620U2 paj;

//BUTTONS
const int buttonPin_A = 26;
const int buttonPin_B = 28;
const int buttonPin_C = 24;
const int buttonPin_D = 22;

int buttonState_A = 0;
int buttonState_B = 0;
int buttonState_C = 0;
int buttonState_D = 0;


// Directions
#define DIR_UP 0
#define DIR_DOWN 1
#define DIR_LEFT 2
#define DIR_RIGHT 3

int resetDelay = 250;

// Game Settings
#define APPLESIZE 10
#define SNAKESIZE 3

// Game State
#define MAX_SPEED 5
#define MIN_SPEED 150
#define SPEED_LOSS 15
int gameSpeed = MIN_SPEED;
int currDirection = DIR_UP;
int timing_bool = 0;
int s = 0;

boolean isGamePaused = false;
boolean isTogglingPause = false;
boolean isGameOver = false;

CRGB targetAppleColour = CRGB(255, 0, 0);  // Red;
```

```
//Score
int numScore = 0;    //starting score
int highScore1 = 0;  // Mode 1 score
int highScore2 = 0;  // Mode 2 score

// base address for highscore memory EEPROM library
int baseAddr = 2000;

// class that represents a point on the matrix. Indexed starting at 0
class Point {
private:
  byte x;
  byte y;
public:
  Point(byte x, byte y) {
    this->x = x;
    this->y = y;
  }
  byte getX() {
    return x;
  }
  byte getY() {
    return y;
  }
  boolean isEqual(int x, int y) {
    return this->x == x && this->y == y;
  }
};


// body of the snake, last element representing the tail, first element
representing the head
LinkedList<Point *> snakePositions = LinkedList<Point *>();
LinkedList<Point *> applePositions = LinkedList<Point *>();
LinkedList<Point *> applePositionsCopy = LinkedList<Point *>();

LinkedList<int> emptyIndices;

// where the apple is located
Point *applePosition;
```

```
// For Style ;)
CRGB appleColor = CRGB(255, 0, 0);
CRGB snakeColor = CRGB(15, 255, 80);
CRGB pausedAppleColor = CRGB(0, 255, 255);
CRGB pausedSnakeColor = CRGB(0, 0, 255);
CRGB emptyColor = CRGB(0, 0, 0);
CRGB solidColor = CRGB(255, 0, 0);
CRGB greenSolidColor = CRGB(0, 255, 0);


// Define arrays for apple colours
CRGB appleColors[] = {
  CRGB(255, 0, 0),    // Red
  CRGB(0, 128, 0),    // Green
  CRGB(0, 0, 255),    // Blue
  CRGB(255, 255, 0),  // Yellow
  CRGB(128, 0, 128)   // Purple

};

// For debug purposes.
String getAppleColorName(CRGB color) {
  if (color == CRGB(255, 0, 0)) {
    return "Red";
  } else if (color == CRGB(0, 128, 0)) {
    return "Green";
  } else if (color == CRGB(0, 0, 255)) {
    return "Blue";
  } else if (color == CRGB(255, 255, 0)) {
    return "Yellow";
  } else if (color == CRGB(128, 0, 128)) {
    return "Purple";
  } else {
    return "Unknown";
  }
}

//OPENCV
void replyToPython() {
  if (newData == true) {
    str = receivedChars;
```

```
    Serial.print("<This just in ... ");
    Serial.print(receivedChars);

    Serial.print("   ");
    Serial.print(millis());
    Serial.print('>');
    // change the state of the LED everytime a reply is sent
    newData = false;
  }
}

void recvWithStartEndMarkers() {
  static boolean recvInProgress = false;
  static byte ndx = 0;
  char startMarker = '<';
  char endMarker = '>';
  char rc;

  while (Serial.available() > 0 && newData == false) {
    rc = Serial.read();

    if (recvInProgress == true) {
      if (rc != endMarker) {
        receivedChars[ndx] = rc;
        ndx++;
        if (ndx >= numChars) {
          ndx = numChars - 1;
        }
      } else {
        receivedChars[ndx] = '\0';  // terminate the string
        recvInProgress = false;
        ndx = 0;
        newData = true;
      }
    }

    else if (rc == startMarker) {
      recvInProgress = true;
    }
  }
}
```

```
//OPENCV END

void setup() {
  Serial.begin(115200);
  Serial.setTimeout(0);
  previousMillis = millis();

  paj.begin();
  paj.setGestureHighRate(true);
  randomSeed(analogRead(0));

  menuSetup();  //set up starting screen button
  //display score setup
  scoreSystem();
  musicSetup();
  sendMP3Command('4');
  timerdisplay(); // Display Timer Screen
  getHighScore();
  // eraseHighScore();

  //setup buttons
  pinMode(buttonPin_A, INPUT);
  pinMode(buttonPin_B, INPUT);
  pinMode(buttonPin_C, INPUT);
  pinMode(buttonPin_D, INPUT);

  // Init gesture sensor
  paj.begin();
  paj.setGestureHighRate(true);

  // pick apple position
  applePosition = getApplePosition();

  resetScore();
  resetSnake();
  resetApple();


  // setup ws2812b leds
  FastLED.addLeds<WS2812, LED_PIN, GRB>(leds, NUM_LEDS);
  FastLED.setBrightness(100);
```

```
  Serial.println("<Arduino is ready>");
}

void gameLoop() {
  unsigned long currentMillis = millis();

  // read the state of the pushbutton value:
  buttonState_A = digitalRead(buttonPin_A);
  buttonState_B = digitalRead(buttonPin_B);
  buttonState_C = digitalRead(buttonPin_C);
  buttonState_D = digitalRead(buttonPin_D);

  //OPENCV
  recvWithStartEndMarkers();
  replyToPython();


  if (currentMillis - previousMillis >= gameSpeed) {
    currDirection = getCurrentDirection();
    previousMillis = currentMillis;

    // get next position
    Point *nextPoint = getNextPosition();

    if (mode1) {
      if (isNextPointValid(nextPoint)) {
        playGame(nextPoint);
      } else {
        setHighScoreOne();
        resetGame(true);
      }

      displayHighScore(1);

    } else {
      // check if we are still valid and continue the game, otherwise
reset it
      if (isNextPointValid(nextPoint) && isCorrectApple(nextPoint)) {
        playGame(nextPoint);  // V2 continue
        timing_bool = 1;
        if (applePositions.size() <= 0) {
          timing_bool = 0;
```

```
          setHighScoreTwo();
          resetGame(false); //V2 Win
          displayHighScore(2);
        }
      } else {
        Serial.println("Reset");
        // playGame(nextPoint);
        timing_bool = 0;
        resetGame(true);  //V2 gameover
        displayHighScore(2);
      }
    }
  }

 }

//to read string input
void readInput() {
  if (Serial.available()) {
    str = Serial.readString();
    Serial.println(str);
  }
}

// always start the game in the same spot
Point *getStartingPosition() {
  return new Point(4, 0);
}

// generate the position of the apple so it is not out of bounds or
within the snake
Point *getApplePosition() {
  Point *snakeStart = getStartingPosition();
  int x, y;

  do {
    x = random(rows);
    y = random(columns);

    // [is not in snake], [no snakestart pos], [same pos another apple],
[same as snake start row]
  } while (snakeContainsPosition(x, y) || snakeStart->isEqual(x, y) ||
```

```
appleContainsPosition(x, y) || x == snakeStart->getX());
  // Serial.print("Coordinates: ");
  // Serial.println(Point(x, y).toString());
  return new Point(x, y);
}

// check if the x y coordinates are covered by a part of the snake
boolean snakeContainsPosition(int x, int y) {
  for (int i = 0; i < snakePositions.size(); i++) {
    if (snakePositions.get(i)->isEqual(x, y)) {
      return true;
    }
  }

  return false;
}

// check if the x y coordinates are covered by another apple
boolean appleContainsPosition(int x, int y) {
  for (int i = 0; i < applePositions.size(); i++) {
    if (applePositions.get(i)->isEqual(x, y)) {
      return true;
    }
  }

  return false;
}

int getCurrentDirection() {
  int dir = currDirection;

  int xPosition = 0;
  int yPosition = 0;

  DFRobot_PAJ7620U2::eGesture_t gesture = paj.getGesture();
  gesInt = gesture;
  // FOR Gesture sensor
  if (gesInt == paj.eGestureUp) {
    dir = DIR_LEFT;
  } else if (gesture == paj.eGestureDown) {
    dir = DIR_RIGHT;
  } else if (gesture == paj.eGestureLeft) {
```

```
    dir = DIR_DOWN;
  } else if (gesture == paj.eGestureRight) {
    dir = DIR_UP;
  } else if (gesture == paj.eGestureForward) {
    // gesInt = 0;
    // directionIndex = 0;
    // currDirection = DIR_UP;
    isGameOver = false;
    // dir = DIR_UP;
  }

  // //FOR STRING INPUT
  if (str == "C") {
    dir = DIR_UP;
  }

  if (str == "D") {
    dir = DIR_RIGHT;
  }

  if (str == "B") {
    dir = DIR_LEFT;
  }

  if (str == "A") {
    dir = DIR_DOWN;
  }


  //FOR button INPUT
  if (buttonState_A == HIGH) {
    dir = DIR_DOWN;
    isGameOver = false;
    Serial.println("s pressed");
  }

  if (buttonState_B == HIGH) {
    dir = DIR_LEFT;
  }

  if (buttonState_C == HIGH) {
    dir = DIR_UP;
```

```
    Serial.println("w pressed");
  }

  if (buttonState_D == HIGH) {
    dir = DIR_RIGHT;
    Serial.println("d pressed");
  }


  //ensure you can't go the direction you just came
  switch (dir) {
    case DIR_UP:
      dir = currDirection == DIR_DOWN ? DIR_DOWN : dir;
      break;
    case DIR_DOWN:
      dir = currDirection == DIR_UP ? DIR_UP : dir;
      break;
    case DIR_LEFT:
      dir = currDirection == DIR_RIGHT ? DIR_RIGHT : dir;
      break;
    case DIR_RIGHT:
      dir = currDirection == DIR_LEFT ? DIR_LEFT : dir;
      break;
    default:
      break;
  }

  return dir;
}

Point *getHead() {
  return snakePositions.get(0);
}

Point *getTail() {
  return snakePositions.get(snakePositions.size() - 1);
}

void addToBeginning(Point *p) {
  snakePositions.add(0, p);
}
```

```
void removeTail() {
  delete (snakePositions.pop());
}

// calculate the next position based on the current head position and
the current direction
Point *getNextPosition() {
  Point *head = getHead();
  switch (currDirection) {
    case DIR_UP:
      return new Point(head->getX(), head->getY() + 1);
    case DIR_DOWN:
      return new Point(head->getX(), head->getY() - 1);
    case DIR_LEFT:
      return new Point(head->getX() - 1, head->getY());
    case DIR_RIGHT:
      return new Point(head->getX() + 1, head->getY());
    default:
      return new Point(-9, -9);
  }
}

// make sure the next point for the head of the snake is in a valid
position
boolean isNextPointValid(Point *p) {
  int x = p->getX();
  int y = p->getY();

  // check if within boundary or if we are in the snake
  if (x < 0 || x >= rows || y < 0 || y >= columns ||
snakeContainsPosition(x, y)) {
    return false;
  }

  return true;
}

boolean isCorrectApple(Point *nextPoint) {
  // if we land on an apple, add score, delete apple and spawn a new one
  for (int i = 0; i <= applePositions.size(); i++) {
    int snakeIndex = getIndexForPoint(nextPoint);
    int appleIndex = getIndexForPoint(applePositions.get(i));
```

```
    //Land on apple
    if (applePositions.get(i)->isEqual(nextPoint->getX(),
nextPoint->getY())) {
      Serial.print("red value:");
      Serial.println(leds[appleIndex].r);
      Serial.print("blue value:");
      Serial.println(leds[appleIndex].b);
      Serial.print("green value:");
      Serial.println(leds[appleIndex].g);
      // Correct colour
      if (leds[appleIndex] == targetAppleColour) {
        emptyIndices.add(appleIndex);
        deleteApple(i);
        // chooseTargetAppleColor();
        Serial.println("appleindex: ");
        Serial.println(appleIndex);
        addScore();
        return true;
      }
      return false;
    }
  }
  return true;
}

// draw the apple
void renderApple() {
  if (mode1) {
    leds[getIndexForPoint(applePosition)] = appleColor;
  } else {
    Point *p;

    for (int i = 0; i < applePositionsCopy.size(); i++) {
      p = applePositionsCopy.get(i);
      int index = getIndexForPoint(p);
      int x = p->getX();
      int y = p->getY();

      // Check if the index matches any value in the emptyIndices linked
list
      bool isIndexEmpty = false;
```

```
      for (int j = 0; j < emptyIndices.size(); j++) {
        if (index == emptyIndices.get(j)) {
          isIndexEmpty = true;
          break;  // No need to continue checking
        }
      }

      if (isIndexEmpty) {
        leds[index] = emptyColor;
      } else {

        int colorIndex = i % (sizeof(appleColors) /
sizeof(appleColors[0]));
        leds[index] = isGamePaused ? pausedAppleColor :
appleColors[colorIndex];

        // leds[index] = isGamePaused ? pausedAppleColor : appleColors[i
% sizeof(appleColors) / sizeof(appleColors[0])];
      }

    }
  }
}

// draw the snake
void renderSnake() {
  Point *p;
  for (int i = 0; i < snakePositions.size(); i++) {
    p = snakePositions.get(i);
    int index = getIndexForPoint(p);
    int x = p->getX();
    int y = p->getY();
    leds[index] = isGamePaused ? pausedSnakeColor : snakeColor;
  }
}

// for a point in the matrix, map it to the index in the string
int getIndexForPoint(Point *p) {
  int x = p->getX();
  int y = p->getY();
  boolean oddRow = x % 2 == 1;
```

```
  // handle serpentine pattern
  if (oddRow) {
    return (x + 1) * columns - y - 1;
  }

  return x * columns + y;
}

void renderEmptyScreen() {
  for (int i = 0; i < NUM_LEDS; i++) {
    leds[i] = emptyColor;
  }
}

void renderSolidScreen(bool gameover) {

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

    if (!gameover) {
      leds[i] = greenSolidColor;
    } else {
      leds[i] = solidColor;
    }
  }
}

void playGame(Point *nextPoint) {
  // clear screen
  renderEmptyScreen();
  renderApple();

  if (mode1) {
    displayScore();  //SCORE SYSTEN DISPLAY LOOP
    if (applePosition->isEqual(nextPoint->getX(), nextPoint->getY())) {
      growSnake(nextPoint);
    } else {
      moveSnake(nextPoint);
    }
  } else {
    displaytimer(timing_bool);

    Serial.print("Target apple colour: ");
```

```
    Serial.println(getAppleColorName(targetAppleColour));
    Serial.println("================================");
    displayColor(targetAppleColour.r, targetAppleColour.g,
targetAppleColour.b);
    moveSnake(nextPoint);
  }

  renderSnake();

  FastLED.show();

  // delay(gameSpeed);
}

void deleteApple(int i) {
  delete (applePositions.remove(i));
  targetAppleColour = chooseTargetAppleColor();
  sendMP3Command('2');
}

CRGB chooseTargetAppleColor() {
  int index = random(applePositions.size());
  Serial.print("index :");
  Serial.println(index);
  int chosenIndex = getIndexForPoint(applePositions.get(index));

  Serial.print("Chosen index: ");
  Serial.println(chosenIndex);
  return leds[chosenIndex];
}

void moveSnake(Point *p) {
  addToBeginning(p);
  removeTail();
}

void growSnake(Point *p) {
  sendMP3Command('2');
  addToBeginning(p);
  resetApple();
  increaseSpeed();
```

```
  //SCORE SYSTEM
  addScore();
}

void increaseSpeed() {
  gameSpeed -= SPEED_LOSS;
  Serial.println(gameSpeed);
  if (gameSpeed <= MAX_SPEED) {
    gameSpeed = 20;  //so that speed will not go to negative or zero
after minusing 15
  }
}

void checkForPause() {
  //boolean isPressedDown = digitalRead(PAUSE_PIN) == 0;
  boolean isPressedDown = str == '0';
  boolean shouldPause = false;
  if (isPressedDown) {
    // we are trying to pause the game and the button is held down
    isTogglingPause = true;
  } else {
    // the pause button is released, so let's trigger a pause
    shouldPause = isTogglingPause;
    isTogglingPause = false;
  }

  if (shouldPause) {
    isGamePaused = !isGamePaused;
  }
}

// initiate a different view
void pauseGame() {
  renderSnake();
  renderApple();
  FastLED.show();
}

// delete the snake and create a new one
void resetSnake() {
  int size = 3;
  if (mode1) {
```

```
    size = 0;
  } else {
    size = SNAKESIZE;
  }
  while (snakePositions.size() > 0) {
    delete (snakePositions.pop());
  }
  snakePositions.add(getStartingPosition());

  // Setting Snake Length

  for (int i = 0; i < size - 1; i++) {
    Point *newPoint;
    newPoint = new Point(getStartingPosition()->getX(),
getStartingPosition()->getY() + i);
    snakePositions.add(0, newPoint);
  }
}

// delete the current position and draw a new apple
void resetApple() {
  delete (applePosition);
  applePosition = getApplePosition();
  if (!mode1) {
    emptyIndices.clear();
    applePositions.clear();
    applePositionsCopy.clear();

    for (int i = 0; i < APPLESIZE; i++) {
      Point *newApple = getApplePosition();
      // Add to the main list
      applePositions.add(0, newApple);
      // Add to the copy list
      applePositionsCopy.add(0, new Point(newApple->getX(),
newApple->getY()));
    }
  }
}

// show an end screen and reset the game state
void resetGame(bool gameover) {
  if (gameover) {
```

```
    sendMP3Command('3');
  } else{
    sendMP3Command('1');
  }
  isGameOver = true;
  resetSnake();
  resetApple();
  resetStr();
  gesInt = 0;
  gameSpeed = MIN_SPEED;
  currDirection = DIR_UP;
  renderSolidScreen(gameover);
  FastLED.show();
  delay(resetDelay);
  renderEmptyScreen();
  FastLED.show();
  delay(resetDelay);
  renderSolidScreen(gameover);
  FastLED.show();
  delay(resetDelay);
  renderEmptyScreen();
  FastLED.show();
  delay(resetDelay);
  renderSolidScreen(gameover);
  FastLED.show();
  delay(resetDelay);
  renderEmptyScreen();
  FastLED.show();
  resetScore();
  resetSnake();
  resetApple();
  renderApple();
  targetAppleColour = CRGB(255, 0, 0);  // Red;
  // targetAppleColour = chooseTargetAppleColor();
  displayColor(0, 0, 0);
  if(!gameover){
    delay(3000);
  }
  sendMP3Command('4');
  setFrameState(0);
}
```

```
void addScore() {
  numScore = numScore + 1;
  Serial.println(numScore);
}

void resetScore() {
  if (numScore != 0) {
    numScore = 0;
  }
}

// EEPROM Codes to store values into memory
void setHighScoreOne() {
  if (mode1 && (highScore1 <= numScore)) {
    highScore1 = numScore;
    EEPROM.put(baseAddr, highScore1);
  }
}

void setHighScoreTwo() {
  if (s <= highScore2 && s > 0) {  // still need to add the condition
where the number of apples is higher
    highScore2 = s;
    EEPROM.put(baseAddr + 4, highScore2);
    Serial.println("s highscore");
    Serial.print(highScore2);
  }
}

// Get value from memory
void getHighScore() {
  // Retrieve Mode 1 high score
  EEPROM.get(baseAddr, highScore1);

  // Retrieve Mode 2 high score
  EEPROM.get(baseAddr + 4, highScore2);
}

// Erase value in memory (for maintenance)
void eraseHighScore() {
  // Mode 1 highscore
  EEPROM.put(baseAddr, 0);
```

```
  // Mode 2 highscore
  EEPROM.put(baseAddr + 4, 1000);
}

void resetStr(){
  str ="";
}
void initGesure(){
  // Init gesture sensor
  paj.begin();
  paj.setGestureHighRate(true);
}
void setGameMode1(bool mode) {
  mode1 = mode;
  resetScore();
  resetSnake();
  resetApple();
}
```

menuDisplay.ino

```
#define NUM_ROWS 28  // Number of rows in the zigzag layout
#define NUM_COLS 20  // Number of columns in the zigzag layoutBLA

#define ENTER_A_PIN 30  // Digital pin ENTER BUTTON A
#define ENTER_B_PIN 32  // Digital pin ENTER BUTTON B
int enterA = 0;
int enterB = 0;

int startFrame = 0;
int frame = 0;

const CRGB BLACK = CRGB(0x000000);
const CRGB CYAN = CRGB(0xa8f0ff);


void menuSetup() {
  pinMode(ENTER_A_PIN, INPUT);  // Set the button pin as input with a
pull-up resistor
  pinMode(ENTER_B_PIN, INPUT);  // Set the button pin as input with a
pull-up resistor
```

```
}



CRGB colors[NUM_ROWS][NUM_COLS] = {
  // Define your first frame (20x2) color codes here...
  // Format: {0xRRGGBB, 0xRRGGBB, ...} for each row
{0x000000, 0x000000, 0x000000, 0x000000, 0x000000, 0x000000, 0x000000,
0x000000, 0x000000, 0x000000, 0x000000, 0x000000, 0x000000, 0x000000,
0x000000, 0x000000, 0x000000, 0x000000, 0x000000, 0x000000},
{0x000000, 0x000000, 0x000000, 0x000000, 0x000000, 0x000000, 0x000000,
0x000000, 0x000000, 0x000000, 0x000000, 0x000000, 0x000000, 0x000000,
0x000000, 0x000000, 0x000000, 0x000000, 0x000000, 0x000000},
{0x000000, 0x000000, 0x000000, 0x000000, 0x000000, 0x000000, 0x000000,
0x000000, 0x000000, 0x000000, 0x000000, 0x000000, 0x000000, 0x000000,
0x000000, 0x000000, 0x000000, 0x000000, 0x000000, 0x000000},
{0x000000, 0x000000, 0x000000, 0xa8f0ff, 0xa8f0ff, 0xa8f0ff, 0x000000,
0x000000, 0x000000, 0x000000, 0x000000, 0x000000, 0xff53ff, 0x000000,
0x000000, 0xa8f0ff, 0xa8f0ff, 0xa8f0ff, 0x000000, 0x000000},
{0x000000, 0x000000, 0xa8f0ff, 0x000000, 0x000000, 0x000000, 0xff53ff,
0xff53ff, 0xff53ff, 0xa8f0ff, 0xa8f0ff, 0xa8f0ff, 0xff53ff, 0x000000,
0xff53ff, 0xa8f0ff, 0x000000, 0x000000, 0x000000, 0x000000},
{0x000000, 0x000000, 0x000000, 0xa8f0ff, 0xa8f0ff, 0x000000, 0xff53ff,
0x000000, 0xff53ff, 0xa8f0ff, 0x000000, 0xa8f0ff, 0xff53ff, 0x000000,
0xff53ff, 0xa8f0ff, 0xa8f0ff, 0xa8f0ff, 0x000000, 0x000000},
{0x000000, 0x000000, 0x000000, 0x000000, 0x000000, 0xa8f0ff, 0xff53ff,
0x000000, 0xff53ff, 0xa8f0ff, 0xa8f0ff, 0xa8f0ff, 0xff53ff, 0xff53ff,
0x000000, 0xa8f0ff, 0x000000, 0x000000, 0x000000, 0x000000},
{0x000000, 0x000000, 0xa8f0ff, 0xa8f0ff, 0xa8f0ff, 0x000000, 0xff53ff,
0x000000, 0xff53ff, 0xa8f0ff, 0x000000, 0xa8f0ff, 0xff53ff, 0x000000,
0xff53ff, 0xa8f0ff, 0xa8f0ff, 0xa8f0ff, 0x000000, 0x000000},
{0x000000, 0x000000, 0x000000, 0x000000, 0x000000, 0x000000, 0x000000,
0x000000, 0x000000, 0x000000, 0x000000, 0x000000, 0x000000, 0x000000,
0x000000, 0x000000, 0x000000, 0x000000, 0x000000, 0x000000},
{0x000000, 0x000000, 0x000000, 0x000000, 0x000000, 0x000000, 0x000000,
0x000000, 0x008000, 0x008000, 0x000000, 0x000000, 0x000000, 0x000000,
0x000000, 0x000000, 0x000000, 0x000000, 0x000000, 0x000000},
{0x000000, 0x000000, 0x000000, 0x000000, 0x000000, 0x000000, 0x008000,
0x008000, 0x008000, 0xffffff, 0x008000, 0x000000, 0x000000, 0x000000,
0x000000, 0x000000, 0x000000, 0x000000, 0x000000, 0x000000},
{0x000000, 0x000000, 0x000000, 0x000000, 0xff0000, 0xff0000, 0xff0000,
0x008000, 0x008000, 0x008000, 0x008000, 0x008000, 0x008000, 0x008000,
```

```
0x008000, 0x008000, 0x000000, 0x000000, 0x000000, 0x000000},
{0x000000, 0x000000, 0x000000, 0x000000, 0x000000, 0x000000, 0x000000,
0x000000, 0x000000, 0x000000, 0x000000, 0x000000, 0x000000, 0x000000,
0x000000, 0x008000, 0x008000, 0x000000, 0x000000, 0x000000},
{0x000000, 0x000000, 0x000000, 0x000000, 0x000000, 0x000000, 0x000000,
0x000000, 0x000000, 0x000000, 0x000000, 0x000000, 0x000000, 0x000000,
0x008000, 0x008000, 0x008000, 0x000000, 0x000000, 0x000000},
{0x000000, 0x000000, 0x000000, 0x000000, 0x000000, 0x000000, 0x000000,
0x000000, 0x000000, 0x000000, 0x000000, 0x000000, 0x008000, 0x008000,
0x008000, 0x000000, 0x000000, 0x000000, 0x000000, 0x000000},
{0x000000, 0x5353ff, 0x5353ff, 0x5353ff, 0x5353ff, 0x5353ff, 0x5353ff,
0x5353ff, 0x5353ff, 0x000000, 0x000000, 0x5353ff, 0x5353ff, 0x5353ff,
0x5353ff, 0x5353ff, 0x5353ff, 0x5353ff, 0x5353ff, 0x000000},
{0x000000, 0x5353ff, 0x000000, 0x000000, 0x000000, 0x000000, 0x000000,
0x000000, 0x5353ff, 0x000000, 0x000000, 0x5353ff, 0x000000, 0x000000,
0x000000, 0x000000, 0x000000, 0xffd966, 0x5353ff, 0x000000},
{0x000000, 0x5353ff, 0x000000, 0xff0000, 0x000000, 0x000000, 0x000000,
0x000000, 0x5353ff, 0x000000, 0x000000, 0x5353ff, 0x000000, 0x000000,
0xff0000, 0x000000, 0xff53ff, 0x000000, 0x5353ff, 0x000000},
{0x000000, 0x5353ff, 0x000000, 0x000000, 0x000000, 0x000000, 0x000000,
0x000000, 0x5353ff, 0x000000, 0x008000, 0x5353ff, 0xa8f0ff, 0x000000,
0x000000, 0x000000, 0x000000, 0x000000, 0x5353ff, 0x000000},
{0x000000, 0x5353ff, 0x5353ff, 0x5353ff, 0x5353ff, 0x5353ff, 0x5353ff,
0x5353ff, 0x5353ff, 0x000000, 0x008000, 0x5353ff, 0x5353ff, 0x5353ff,
0x5353ff, 0x5353ff, 0x5353ff, 0x5353ff, 0x5353ff, 0x000000},
{0x000000, 0x000000, 0x000000, 0x000000, 0x000000, 0x000000, 0x000000,
0x000000, 0x000000, 0x008000, 0x008000, 0x008000, 0x000000, 0x000000,
0x000000, 0x000000, 0x000000, 0x000000, 0x000000, 0x000000},
{0x000000, 0x000000, 0x000000, 0x000000, 0xffd966, 0x000000, 0x000000,
0x000000, 0x000000, 0x008000, 0x008000, 0x008000, 0x000000, 0xffd966,
0xffd966, 0x000000, 0x000000, 0x000000, 0x000000, 0x000000},
{0x000000, 0x000000, 0x000000, 0xffd966, 0x000000, 0xffd966, 0x000000,
0x000000, 0x008000, 0x008000, 0x008000, 0x000000, 0x000000, 0xffd966,
0x000000, 0xffd966, 0x000000, 0x000000, 0x000000, 0x000000},
{0x000000, 0x000000, 0x000000, 0xffd966, 0xffd966, 0xffd966, 0x000000,
0x008000, 0x008000, 0x008000, 0x000000, 0x000000, 0x000000, 0xffd966,
0xffd966, 0x000000, 0x000000, 0x000000, 0x000000, 0x000000},
{0x000000, 0x000000, 0x000000, 0xffd966, 0x000000, 0xffd966, 0x000000,
0x008000, 0x008000, 0x000000, 0x000000, 0x000000, 0x000000, 0xffd966,
0x000000, 0xffd966, 0x000000, 0x000000, 0x000000, 0x000000},
{0x000000, 0x000000, 0x000000, 0xffd966, 0x000000, 0xffd966, 0x000000,
0x008000, 0x008000, 0x008000, 0x000000, 0x000000, 0x000000, 0xffd966,
```

```
0xffd966, 0x000000, 0x000000, 0x000000, 0x000000, 0x000000},
{0x000000, 0x000000, 0x000000, 0x000000, 0x000000, 0x000000, 0x000000,
0x008000, 0x008000, 0x008000, 0x008000, 0x008000, 0x000000, 0x000000,
0x000000, 0x000000, 0x000000, 0x000000, 0x000000, 0x000000},
{0x000000, 0x000000, 0x000000, 0x000000, 0x000000, 0x000000, 0x000000,
0x000000, 0x000000, 0x008000, 0x008000, 0x008000, 0x008000, 0x008000,
0x008000, 0x008000, 0x008000, 0x008000, 0x008000, 0x008000},
  // Define hex codes for the first frame for all rows and columns
};

void loop() {
  frame = getFrameState();
  // SNAKE VERSION 1
  if (frame == 1 || frame == 2) {
    gameLoop();
  }

  //START MENU FRAME
  if (frame == 0) {
    // renderSolidScreen();
    initGesure();
    enterA = digitalRead(ENTER_A_PIN);
    enterB = digitalRead(ENTER_B_PIN);
    if (enterA) {
      resetStr();
      setFrameState(1);
      // sendMP3Command('4');
      sendMP3Command('5');
      setGameMode1(true);
      renderEmptyScreen();
    } else if (enterB) {
      resetStr();
      setFrameState(2);
      // sendMP3Command('4');
      sendMP3Command('5');
      setGameMode1(false);
      renderEmptyScreen();

    }

    // Display the current frame on the LED strip
    for (int row = 0; row < NUM_ROWS; row++) {
```

```
      if (row % 2 == 0) {
        // Even rows (left to right)
        for (int col = 0; col < NUM_COLS; col++) {
          leds[row * NUM_COLS + col] = colors[row][col] ;
        }
      } else {
        // Odd rows (right to left)
        for (int col = 0; col < NUM_COLS; col++) {
          leds[row * NUM_COLS + (NUM_COLS - 1 - col)] =  colors[row][col]
;
        }
      }
    }
  }

  FastLED.show();
}
void setFrameState(int i) {
  frame = i;
}

int getFrameState() {

  return frame;
}
```

scoredisplay.ino

```
#include <Adafruit_GFX.h>
#include <Adafruit_NeoMatrix.h>
#include <Adafruit_NeoPixel.h>
#ifndef PSTR
#define PSTR  // Make Arduino Due happy
#endif

#define PIN 49

int x = 0;
int pass = 0;

Adafruit_NeoMatrix matrix = Adafruit_NeoMatrix(10, 7, PIN,
                                               NEO_MATRIX_TOP +
```

```
NEO_MATRIX_LEFT + NEO_MATRIX_ROWS + NEO_MATRIX_ZIGZAG,
                                                NEO_GRB + NEO_KHZ800);

const uint16_t scoreColors[] = {
  matrix.Color(255, 0, 0), matrix.Color(0, 255, 0), matrix.Color(0, 0,
255)
};

void scoreSystem() {
  matrix.begin();
  matrix.setTextWrap(false);
  matrix.setBrightness(40);
  matrix.setTextColor(scoreColors[0]);
  // matrix.setTextSize(1); //old static score
  matrix.setTextColor(colors[0]);
}

void displayScore() {

  //OLD STATIC SCORE
  // matrix.setCursor(0, 0);
  // uint32_t blackColor = matrix.Color(0, 0, 0);  //to clear the screen
after each number change
  // matrix.fillScreen(blackColor);
  // matrix.print(numScore);
  // matrix.show();
  // delay(100);

  matrix.fillScreen(0);

  if(numScore > 9){
     matrix.setCursor(x, 0);
  } else {
    matrix.setCursor(3,0);
  }
  matrix.print(numScore);
  if (x < 0) {
    x++;
  } else {
    x--;
  }
  //if(x > 10) {
```

```
  // x = -10;
  if (pass >= 3) pass = 0;
  matrix.setTextColor(colors[pass]);
  matrix.show();
  // delay(400);

}

// Display Snake V2 Next Color
void displayColor(int r, int g, int b) {
  uint32_t color = matrix.Color(r, g, b);
  matrix.fillScreen(color);
  matrix.show();
}
```

timerDisplay.ino

```
#define PIN 51
#define PIN_NUMBER 51

Adafruit_NeoMatrix timermatrix = Adafruit_NeoMatrix(17, 7, PIN,
  NEO_MATRIX_TOP     + NEO_MATRIX_LEFT +
  NEO_MATRIX_ROWS + NEO_MATRIX_ZIGZAG,
  NEO_GRB            + NEO_KHZ800);

bool is_timing = 0; // // 1 means we are timing the interval between
triggers; 0 means we are not
unsigned long start_time;   // stores the start of the timing interval


void timerdisplay() {
  timermatrix.begin();
  timermatrix.setTextWrap(false);
  timermatrix.setBrightness(40);
  timermatrix.setTextColor(scoreColors[0]);
  timermatrix.setTextSize(1);
}

void displaytimer(int timing_bool) {
  timermatrix.setCursor(0, 0);
  uint32_t blackColor = timermatrix.Color(0, 0, 0); //to clear the screen
after each number change
```

```
  timermatrix.fillScreen(blackColor);
  timermatrix.print(countUpTimer(timing_bool));
  timermatrix.show();
  delay(100);
}

void displayHighScore(int mode) {
  // Retrieve highscore in memory
  getHighScore();

  timermatrix.setCursor(0, 0);
  uint32_t blackColor = timermatrix.Color(0, 0, 0); //to clear the screen
after each number change
  timermatrix.fillScreen(blackColor);
  if (mode == 1) {
    timermatrix.print(highScore1);

  } else if (mode == 2) {
      if (highScore2 >= 1000) {
        timermatrix.print(0);
      } else {
        timermatrix.print(highScore2);
      }
  }

  timermatrix.show();
  delay(100);
}

int countUpTimer(int timing_bool) {

  is_timing = timing_bool;

  if (!is_timing && !digitalRead(PIN_NUMBER)) {// trigger the start of
the timing interval
    is_timing = 1;            // we are now timing the interval
    start_time = millis();   // record the current time as the start of
the interval
  }
  if (is_timing) {
    unsigned long elapsed_time = (millis() - start_time) / 1000;
    byte seconds = elapsed_time;
```

```
    // byte seconds = elapsed_time % 60;
    // byte hours = (elapsed_time / 3600) % 100;
    // byte minutes = (elapsed_time / 60) % 60;

    s = seconds;
    // int m = minutes;

    if (elapsed_time > 999 && !digitalRead(PIN_NUMBER))is_timing = 0;

    return s;
  }

  return 0;
}
```

music.ino

```
/*
modified on Sep 7, 2020
by cefaloide Examples https://github.com/cefaloide/ArduinoSerialMP3Player
Home
*/

#include <SoftwareSerial.h>

#define ARDUINO_RX 5  //should connect to TX of the Serial MP3 Player
module **p
#define ARDUINO_TX 6  //connect to RX of the module

SoftwareSerial mp3(ARDUINO_RX, ARDUINO_TX);
//#define mp3 Serial3    // Connect the MP3 Serial Player to the Arduino
MEGA Serial3 (14 TX3 -> RX, 15 RX3 -> TX)

static int8_t Send_buf[8] = { 0 };  // Buffer for Send commands.  //
BETTER LOCALLY
static uint8_t ansbuf[10] = { 0 };  // Buffer for the answers.    //
BETTER LOCALLY

String mp3Answer;  // Answer from the MP3.

/*******4***** Command byte **************************/
#define CMD_NEXT_SONG 0X01  // Play next song.
```

```
#define CMD_PREV_SONG 0X02  // Play previous song.
#define CMD_PLAY_W_INDEX 0X03
#define CMD_VOLUME_UP 0X04
#define CMD_VOLUME_DOWN 0X05
#define CMD_SET_VOLUME 0X06

#define CMD_SNG_CYCL_PLAY 0X08  // Single Cycle Play.
#define CMD_SEL_DEV 0X09
#define CMD_SLEEP_MODE 0X0A
#define CMD_WAKE_UP 0X0B
#define CMD_RESET 0X0C
#define CMD_PLAY 0X0D
#define CMD_PAUSE 0X0E
#define CMD_PLAY_FOLDER_FILE 0X0F

#define CMD_STOP_PLAY 0X16
#define CMD_FOLDER_CYCLE 0X17
#define CMD_SHUFFLE_PLAY 0x18   //
#define CMD_SET_SNGL_CYCL 0X19  // Set single cycle.

#define CMD_SET_DAC 0X1A
#define DAC_ON 0X00
#define DAC_OFF 0X01

#define CMD_PLAY_W_VOL 0X22
#define CMD_PLAYING_N 0x4C
#define CMD_QUERY_STATUS 0x42
#define CMD_QUERY_VOLUME 0x43
#define CMD_QUERY_FLDR_TRACKS 0x4e
#define CMD_QUERY_TOT_TRACKS 0x48
#define CMD_QUERY_FLDR_COUNT 0x4f

/************ Opitons **************************/
#define DEV_TF 0X02


/*****************************************************************************/

void musicSetup() {
  // Serial.begin(9600);
  mp3.begin(9600);
  delay(500);
```

```
  sendCommand(CMD_SEL_DEV, DEV_TF);
  delay(500);
}


// void loop()
// {
//   char c = ' ';

//   // If there a char on Serial call sendMP3Command to sendCommand
//   if ( Serial.available() )
//   {
//     c = Serial.read();
//     sendMP3Command(c);
//   }

//   // Check for the answer.
//   if (mp3.available())
//   {
//     Serial.println(decodeMP3Answer());
//   }
//   delay(100);
// }


/************************************************************************
********/
/*Function sendMP3Command: seek for a 'c' command and send it to MP3  */
/*Parameter: c. Code for the MP3 Command, 'h' for help.
*/
/*Return:  void
*/

void sendMP3Command(char c) {
  switch (c) {
    case '?':
    case 'h':
      Serial.println("HELP  ");
      Serial.println(" p = Play");
      Serial.println(" P = Pause");
      Serial.println(" > = Next");
```

```
      Serial.println(" < = Previous");
      Serial.println(" + = Volume UP");
      Serial.println(" - = Volume DOWN");
      Serial.println(" c = Query current file");
      Serial.println(" q = Query status");
      Serial.println(" v = Query volume");
      Serial.println(" x = Query folder count");
      Serial.println(" t = Query total file count");
      Serial.println(" 1 = Play folder 1");
      Serial.println(" 2 = Play folder 2");
      Serial.println(" 3 = Play folder 3");
      Serial.println(" 4 = Play folder 4");
      Serial.println(" 5 = Play folder 5");
      Serial.println(" S = Sleep");
      Serial.println(" W = Wake up");
      Serial.println(" r = Reset");
      break;


    case 'p':
      Serial.println("Play ");
      sendCommand(CMD_PLAY, 0);
      //sendCommand(CMD_FOLDER_CYCLE, 0x0101); //first file
      //sendCommand(CMD_FOLDER_CYCLE, 0x0201); //second file
      break;

    case 'P':
      Serial.println("Pause");
      sendCommand(CMD_PAUSE, 0);
      break;

    case '>':
      Serial.println("Next");
      sendCommand(CMD_NEXT_SONG, 0);
      sendCommand(CMD_PLAYING_N, 0x0000);  // ask for the number of file
is playing
      break;

    case '<':
      Serial.println("Previous");
      sendCommand(CMD_PREV_SONG, 0);
      sendCommand(CMD_PLAYING_N, 0x0000);  // ask for the number of file
```

```
is playing
      break;

    case '+':
      Serial.println("Volume Up");
      sendCommand(CMD_VOLUME_UP, 0);
      break;

    case '-':
      Serial.println("Volume Down");
      sendCommand(CMD_VOLUME_DOWN, 0);
      break;

    case 'c':
      Serial.println("Query current file");
      sendCommand(CMD_PLAYING_N, 0);
      break;

    case 'q':
      Serial.println("Query status");
      sendCommand(CMD_QUERY_STATUS, 0);
      break;

    case 'v':
      Serial.println("Query volume");
      sendCommand(CMD_QUERY_VOLUME, 0);
      break;

    case 'x':
      Serial.println("Query folder count");
      sendCommand(CMD_QUERY_FLDR_COUNT, 0);
      break;

    case 't':
      Serial.println("Query total file count");
      sendCommand(CMD_QUERY_TOT_TRACKS, 0);
      break;

    case '1':

      Serial.println("Play folder 1");
      //sendCommand(CMD_FOLDER_CYCLE, 0x01);
```

```
      sendCommand(CMD_PLAY_FOLDER_FILE, 0x0101);
      break;


    case '2':
      Serial.println("Play folder 2");
      //sendCommand(CMD_FOLDER_CYCLE, 0x02); //should not put '01' behind
each 0 x 02
      sendCommand(CMD_PLAY_FOLDER_FILE, 0x0202);  // Play file in folder
2 with index
      break;

    case '3':
      Serial.println("Play folder 3");
      //sendCommand(CMD_FOLDER_CYCLE, 0x03);
      sendCommand(CMD_PLAY_FOLDER_FILE, 0x0303);  // Play file in folder
3 with index 003
      break;

    case '4':
      Serial.println("Play folder 4");
      sendCommand(CMD_SNG_CYCL_PLAY, 0x0404);  // Play file in folder 4
with index 004 on loop
      break;

    case '5':
      Serial.println("Play folder 5");
      sendCommand(CMD_STOP_PLAY, 0);

      break;
    case '6':
      Serial.println("Play folder 6");
      sendCommand(CMD_PLAY_FOLDER_FILE, 0x0606);

      break;

    case 'S':
      Serial.println("Sleep");
      sendCommand(CMD_SLEEP_MODE, 0x00);
      break;

    case 'W':
```

```
      Serial.println("Wake up");
      sendCommand(CMD_WAKE_UP, 0x00);
      break;

    case 'r':
      Serial.println("Reset");
      sendCommand(CMD_RESET, 0x00);
      break;
  }
}



/*********************************************************************
********/
/*Function decodeMP3Answer: Decode MP3 answer.
*/
/*Parameter:-void
*/
/*Return: The                                                    */

String decodeMP3Answer() {
  String decodedMP3Answer = "";

  decodedMP3Answer += sanswer();

  switch (ansbuf[3]) {
    case 0x3A:
      decodedMP3Answer += " -> Memory card inserted.";
      break;

    case 0x3D:
      decodedMP3Answer += " -> Completed play num " + String(ansbuf[6],
DEC);  //**
      break;

    case 0x40:
      decodedMP3Answer += " -> Error";
      break;

    case 0x41:
      decodedMP3Answer += " -> Data recived correctly. ";
```

```
      break;

    case 0x42:
      decodedMP3Answer += " -> Status playing: " + String(ansbuf[6],
DEC);
      break;

    case 0x48:
      decodedMP3Answer += " -> File count: " + String(ansbuf[6], DEC);
      break;

    case 0x4C:
      decodedMP3Answer += " -> Playing: " + String(ansbuf[6], DEC);
      break;

    case 0x4E:
      decodedMP3Answer += " -> Folder file count: " + String(ansbuf[6],
DEC);
      break;

    case 0x4F:
      decodedMP3Answer += " -> Folder count: " + String(ansbuf[6], DEC);
      break;
  }

  return decodedMP3Answer;
}




/************************************************************************
********/
/*Function: Send command to the MP3
*/
/*Parameter:-int8_t command
*/
/*Parameter:-int16_ dat  parameter for the command
*/
```

```
void sendCommand(int8_t command, int16_t dat) {
  delay(20);
  Send_buf[0] = 0x7e;                 //
  Send_buf[1] = 0xff;                 //
  Send_buf[2] = 0x06;                 // Len
  Send_buf[3] = command;              //
  Send_buf[4] = 0x01;                 // 0x00 NO, 0x01 feedback
  Send_buf[5] = (int8_t)(dat >> 8);   //datah
  Send_buf[6] = (int8_t)(dat);        //datal
  Send_buf[7] = 0xef;                 //
  // Serial.print("Sending: ");
  for (uint8_t i = 0; i < 8; i++) {
    mp3.write(Send_buf[i]);
    // Serial.print(sbyte2hex(Send_buf[i]));
  }
  Serial.println();
}



/*********************************************************************
********/
/*Function: sbyte2hex. Returns a byte data in HEX format.
*/
/*Parameter:- uint8_t b. Byte to convert to HEX.
*/
/*Return: String
*/


String sbyte2hex(uint8_t b) {
  String shex;

  shex = "0X";

  if (b < 16) shex += "0";
  shex += String(b, HEX);
  shex += " ";
  return shex;
}
```

```
/*************************************************************************
********/
/*Function: sanswer. Returns a String answer from mp3 UART module.
*/
/*Parameter:- uint8_t b. void.
*/
/*Return: String. If the answer is well formated answer.
*/

String sanswer(void) {
  uint8_t i = 0;
  String mp3answer = "";

  // Get only 10 Bytes
  while (mp3.available() && (i < 10)) {
    uint8_t b = mp3.read();
    ansbuf[i] = b;
    i++;

    mp3answer += sbyte2hex(b);
  }

  // if the answer format is correct.
  if ((ansbuf[0] == 0x7E) && (ansbuf[9] == 0xEF)) {
    return mp3answer;
  }

  return "???: " + mp3answer;
}
```