Overview

In this tutorial, we will take you into a world of creativity and fun in electronic making, and together we will create a classic Snake game with our own hands. By skillfully combining an OLED screen and a joystick sensor, you will experience firsthand how to combine simple hardware devices with programming to create a unique maker experience.

Component Required:

(1) x Elegoo ESP32

(1) x I2C OLED Display

(1) x Joystick module

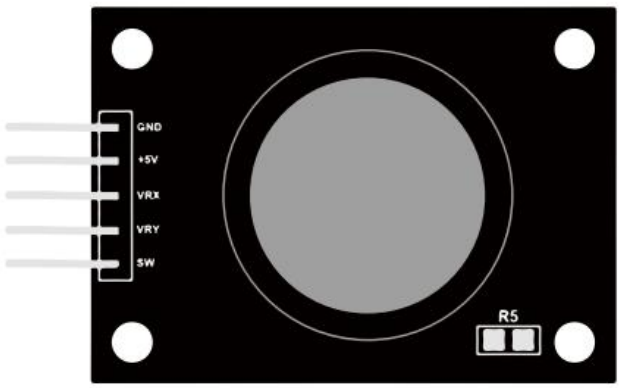
Component Introduction

Joystick

The module has 5 pins: VCC, Ground, X, Y, Key. Note that the labels on yours may be slightly different, depending on where you got the module from. The thumb stick is analog and should provide more accurate readings than simple 'directional' joysticks that use some forms of buttons, or mechanical switches. Additionally, you can press the joystick down (rather hard on mine) to activate a 'press to select 'push- button.

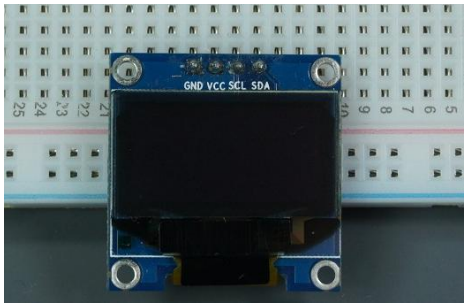
We have to use analog Arduino pins to read the data from the X/Y pins, and a digital pin to read the button. The Key pin is connected to ground, when the joystick is pressed down, and is floating otherwise. To get stable readings from the Key /Select pin, it needs to be connected to VCC via a pull-up resistor.

The built in resistors on the Arduino digital pins can be used. For a tutorial on how to activate the pull-up resistors for Arduino pins, configured as inputs.For example: **pinMode(SW_pin, INPUT);**



I2C OLED Display

The organic light-emitting diode (OLED) display that we'll use in this tutorial is the SSD1306 model: a monocolor, 0.96-inch display with 128×64 pixels as shown in the following figure.



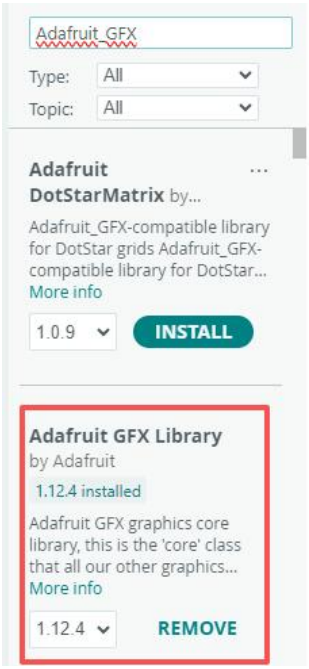
The OLED display doesn't require backlight, which results in a very nice contrast in dark environments. Additionally, its pixels consume energy only when they are on, so the OLED display consumes less power when compared with other displays.

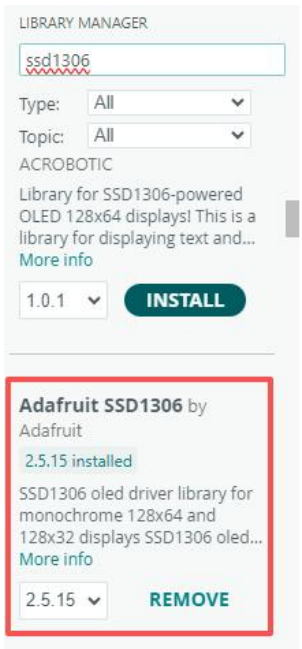
The model we're using here has only four pins and communicates with the ESP32 using I2C communication protocol. There are models that come with an extra RESET pin. There are also other OLED displays that communicate using SPI communication.

Because the OLED display uses I2C communication protocol, wiring is very simple. You just need to connect to the ESP32 I2C pins as D21(SDA)、D22(SCL)

To control the OLED display you need the adafruit_SSD1306.h and the adafruit_GFX.h libraries. Follow the next instructions to install those libraries.

1. Open your Arduino IDE and go to Sketch > Include Library > Manage Libraries. The Library Manager should open.
2. Type "SSD1306" in the search box and install the SSD1306 library from Adafruit.





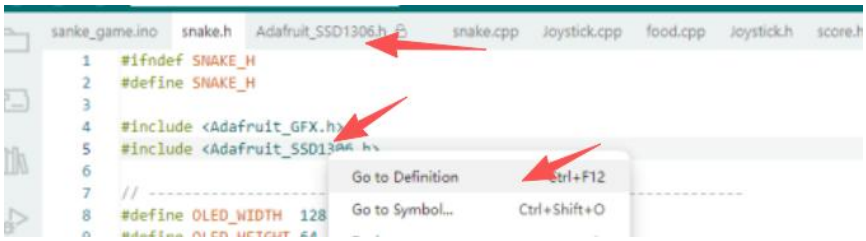
Tips for writing text using these libraries

Here's some functions that will help you handle the OLED display library to write text or draw simple graphics.

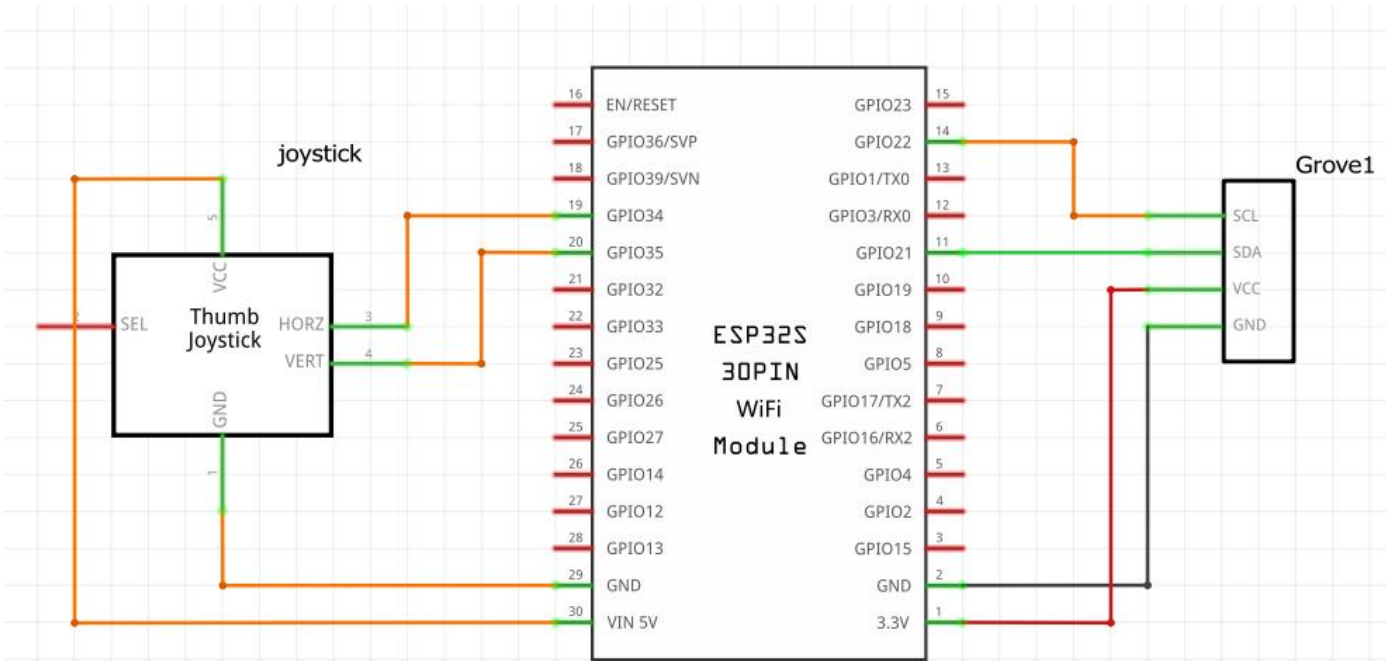
- ■display.clearDisplay() – all pixels are off
- ■display.drawPixel(x,y, color) – plot a pixel in the x,y coordinates
- ■display.setTextSize(n) – set the font size, supports sizes from 1 to 8
- ■display.setCursor(x,y) – set the coordinates to start writing text
- ■display.print("message") – print the characters at location x,y
- ■display.display() – call this method for the changes to make effect

If you want to learn more about the functions available in these two libraries, you can follow these steps:

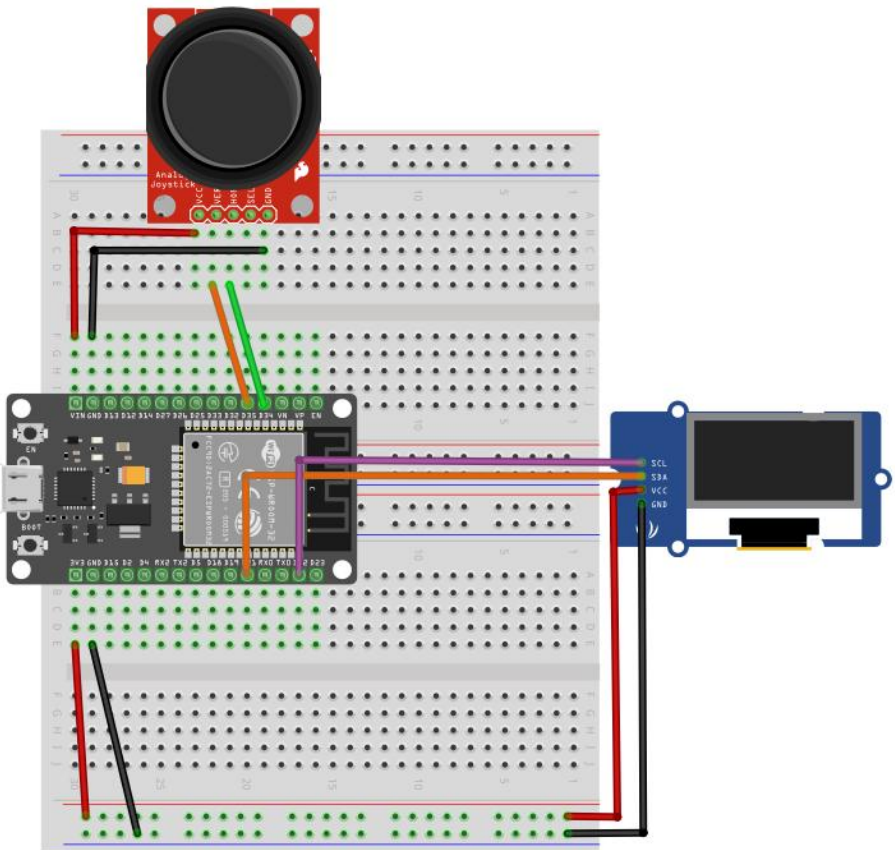
Locate the library definition: Find the position where the .h file is included. Hover the mouse over the library name, then right-click and select "Go to Definition" or "Open Declaration". This will allow you to view all the function declarations and detailed descriptions in the library file. Carefully read the comments to understand the parameter types, return values, and usage examples of each function. For commonly used functions, it is recommended to refer to the official documentation for a better understanding, ensuring that parameters are correctly passed when calling the functions. Additionally, referring to the implementation logic in the example code can help you quickly grasp the calling process and key points of the core methods.



Deepen your understanding of functions in example code: While reading the example code, if you encounter a function that you do not understand, you can hover the mouse over the function to view its specific parameters and definition location. Additionally, you can use the previously mentioned method, right-click the function and select "Go to Definition," to see its detailed usage and implementation logic.



Connection Schematic



Wiring diagram

Program Logic & Call Relationship ：

I. Core Program Architecture

This program adopts a modular design pattern, split into 5 core functional modules. Each module decouples interface declarations (via header files) and logic implementation (via .cpp files), with the main program acting as the unified scheduler. The architecture features clear boundaries, low coupling, and high maintainability.

Module Division & Core Responsibilities

Module	Core Files	Primary Responsibilities
Main Program Module	SnakeGame.ino	Program entry point (initialization via setup(), cyclic execution via loop()), initialization of global objects/variables
Joystick Module	Joystick.h/Joystick.cpp	Hardware initialization for joystick, reading analog input to update snake movement direction
Snake Module	Snake.h/Snake.cpp	Core snake logic (movement, reset, boundary detection, rendering), management of global macros/constants
Food Module	Food.h/Food.cpp	Food spawning (avoiding overlap with snake body), food rendering, collision detection for food consumption
Score Module	Score.h/Score.cpp	Score lifecycle management (initialization, increment, reset), score rendering on OLED

II. Detailed Logic of Key Modules

1. Joystick Module (Joystick.cpp)

(1) initJoystick():

- Sets JOY_X_PIN (GPIO34) and JOY_Y_PIN (GPIO35) to INPUT mode.
- Serial prints: "Joystick initialized successfully" for debug confirmation.

(2) readJoystick():

- Reads analog values (0-4095, 12-bit ADC of ESP32) from X/Y pins.
- Filters drift using DEAD_ZONE=600 (ignores small value fluctuations).
- Updates current_dir (global variable) with reverse movement restriction:
  - Up (0) → cannot switch directly to Down (2)
  - Right (1) → cannot switch directly to Left (3)
  - Down (2) → cannot switch directly to Up (0)
  - Left (3) → cannot switch directly to Right (1)

2. Snake Module (Snake.cpp)

(1) moveSnake():

- Skips execution if current_dir = -1 (no direction input).
- Updates snake body coordinates (tail to head): each segment inherits the position of the previous segment.
- Moves snake head by MOVE_STEP=1 pixel based on current_dir:
  - Up (0): snake_y[0] -= MOVE_STEP
  - Right (1): snake_x[0] += MOVE_STEP
  - Down (2): snake_y[0] += MOVE_STEP
  - Left (3): snake_x[0] -= MOVE_STEP

(2) checkSnakeOverBoundary():

- Boundary criteria: snake_x[0] < 0 / snake_x[0] > MAX_SNAKE_X (124) / snake_y[0] < 0 / snake_y[0] > MAX_SNAKE_Y (60).
- Returns true if out of bounds (triggers game reset), false otherwise.

(3) resetSnake():

- Moves snake head to screen center: (OLED_WIDTH - SNAKE_SIZE)/2 = 62, (OLED_HEIGHT - SNAKE_SIZE)/2 = 30.
- Resets snake_length = 1 (only head remains) and current_dir = -1 (stationary).
- Resets isShow = true (head blink state) and prints debug info via serial.

(4) drawSnake():

- Renders snake body (segments 1 to snake_length-1) with SNAKE_BODY_COLOR (no blink).
- Renders snake head (segment 0) with SNAKE_HEAD_COLOR, toggling visibility via isShow.

3.Food Module (Food.cpp)

(1) generateFood()

- Coordinate alignment ensures food position is a multiple of 4 (matching snake size).
- Loops until non-overlapping coordinates are generated.

(2) checkFoodCollision():

- Collision criteria: Snake head overlaps with food by more than 50% (center overlap for 4x4 pixels).
- If collided:
  - Increments snake_length (capped at MAX_LENGTH=30).
  - Calculates new tail coordinates (opposite to current movement direction).
  - Assigns new coordinates to snake_x[snake_length]/snake_y[snake_length].
  - Returns true (triggers food regeneration and score increment).
- Returns false if no collision.

(3) drawFood():

- Renders food as a circle at (food_x + FOOD_SIZE/2, food_y + FOOD_SIZE/2) with radius FOOD_SIZE/2.

4. Score Module (Score.cpp)

- ●Score Rules:
  - ■ Initial score: 0 points.
  - ■ +1 point per food eaten (adjustable via addScore(val) parameter).
  - ■ Score resets to 0 when snake head is out of bounds.
- ●Rendering:
  - ■ Score is rendered at the top-left corner (2,2) with 8x8 pixel font (size 1), ensuring it is drawn first to avoid being covered by snake/food.

III. Global Variables & Macro Definitions

1. Key Global Variables (Declared in Snake.h, Initialized in SnakeGame.ino)

Variable Name	Type	Purpose	Initial/Range Value
snake_x[]	int[]	X coordinates of snake segments (0 = head, length-1 = tail)	Length: MAX_LENGTH=30, Center (62)
snake_y[]	int[]	Y coordinates of snake segments	Length: MAX_LENGTH=30, Center (30)
snake_length	int	Current length of the snake	1 (head only) → Max 30
current_dir	int	Snake movement direction (-1=None, 0=Up, 1=Right, 2=Down, 3=Left)	-1 (stationary)
isShow	bool	Blink state of snake head (visible/hidden)	true (visible)
food_x/food_y	int	Coordinates of food	Multiple of 4, within valid bounds
current_score	int	Current game score (declared in Score.h)	0 → Incremental

2. Critical Macro Definitions (Defined in Snake.h)

Macro Name	Value/Expression	Purpose
OLED_WIDTH/OLED_HEIGHT	128/64	Resolution of OLED display
JOY_X_PIN/JOY_Y_PIN	34/35	GPIO pins for joystick X/Y axis (analog input)
DEAD_ZONE	600	Dead zone for joystick (filters analog input drift)
MOVE_STEP	1	Pixel step size for snake movement per frame
MOVE_SPEED	50	Frame delay (ms) - balances smoothness and stability
SNAKE_SIZE/FOOD_SIZE	4	Size of snake head/body/food (4x4 pixels)
MAX_SNAKE_X/MAX_SNAKE_Y	124/60	Maximum valid coordinates for snake head (prevents partial out-of-bounds)
MAX_LENGTH	30	Maximum snake length (prevents array out-of-bounds errors)

IV. Key Rules & Constraints

1.Movement Rules:

- The snake cannot move in the reverse direction (e.g., cannot switch from Up to Down directly).
- No hard boundary restriction (snake head can move out of screen bounds).

2.Boundary Reset Rule:

- If any part of the snake head exceeds the screen (x<0/x>124/y<0/y>60), the snake, score, and food are immediately reset to initial states.

3.Food Consumption Rule:

- A collision (50%+ overlap) between snake head and food triggers length increment (+1) and score increment (+1).

4.Hardware Constraints:

- OLED display resolution: 128x64 pixels (I2C communication via GPIO21/SDA, GPIO22/SCL).
- Joystick input: Analog values (0-4095) from ESP32 12-bit ADC.

5.Performance Constraints:

- Maximum snake length: 30 segments (prevents array overflow).
- Frame delay: 50ms (balances game smoothness and CPU utilization)