專案名稱：ESP#@GAMEWATCH

目的：一個可以看時間、天氣跟玩貪吃蛇的小玩具

使用元件：gc9a01（一個圓形的手錶螢幕）

：滾輪編碼器

使用工具：Thonny(Python IDE)、claude(ai assist)、gitkraken(git工具)

程式：

```
class RotaryEncoder:
    """滾輪編碼器類別 - 修正版本"""
    def __init__(self, clk_pin, dt_pin, sw_pin=None):
        self.clk = Pin(clk_pin, Pin.IN, Pin.PULL_UP)
        self.dt = Pin(dt_pin, Pin.IN, Pin.PULL_UP)
        self.sw = Pin(sw_pin, Pin.IN, Pin.PULL_UP) if sw_pin else
None

        self.clk_last = self.clk.value()
        self.dt_last = self.dt.value()
        self.pending_rotation = 0  # 使用單一變數而非緩衝區
        self.button_pressed = False
        self.last_rotation_time = 0
        self.debounce_time = 50  # 縮短防抖時間
        self.last_button_time = 0

    def get_raw_states(self):
        """獲取原始腳位狀態進行調試"""
        return self.clk.value(), self.dt.value()

    def update(self):
        """改進的編碼器檢測"""
        current_time = time.ticks_ms()

        # 讀取當前狀態
        clk_current = self.clk.value()
        dt_current = self.dt.value()

        # 檢測 CLK 下降沿
        if clk_current == 0 and self.clk_last == 1:
            # 檢查防抖時間
            if time.ticks_diff(current_time,
self.last_rotation_time) > self.debounce_time:
                # 在 CLK 下降沿時檢查 DT 狀態
                if dt_current == 1:
                    self.pending_rotation = 1  # 順時針
                    print(f"編碼器: 順時針旋轉")
                else:
                    self.pending_rotation = -1  # 逆時針
                    print(f"編碼器: 逆時針旋轉")

                self.last_rotation_time = current_time

        # 更新歷史狀態
        self.clk_last = clk_current
        self.dt_last = dt_current

        # 檢查按鈕
        if self.sw and self.sw.value() == 0:
            if time.ticks_diff(current_time, 
self.last_button_time) > 200:
                self.button_pressed = True
                self.last_button_time = current_time
                print("按鈕按下")

    def get_rotation(self):
        """獲取並清除旋轉值"""
        self.update()

        if self.pending_rotation != 0:
            rotation = self.pending_rotation
            self.pending_rotation = 0  # 清除已讀取的值
            return rotation
        return 0

    def peek_rotation(self):
        """查看但不清除旋轉值"""
        self.update()
        return self.pending_rotation

    def clear_rotation(self):
        """清除累積的旋轉值"""
        self.pending_rotation = 0

    def is_button_pressed(self):
        """檢查按鈕"""
        self.update()
        if self.button_pressed:
            self.button_pressed = False
            return True
        return False

class WeatherAPI:
    """中央氣象局 API 處理類別"""
    def __init__(self, api_key):
        self.api_key = api_key
        self.base_url = "https://opendata.cwa.gov.tw/api/v1/rest/
datastore/F-C0032-001"
        self.weather_data = None
        self.last_update = 0
        self.update_interval = 1800000  # 30分鐘更新一次

    def connect_wifi(self, ssid, password):
        """連接 WiFi"""
        wlan = network.WLAN(network.STA_IF)
        wlan.active(True)

        if not wlan.isconnected():
            print('連接 WiFi...')
            wlan.connect(ssid, password)

            timeout = 10
            while not wlan.isconnected() and timeout > 0:
                time.sleep(1)
                timeout -= 1

            if wlan.isconnected():
                print('WiFi 連接成功')
                print('IP:', wlan.ifconfig()[0])
                return True
            else:
                print('WiFi 連接失敗')
                return False
        return True

    def sync_time(self):
        """同步網路時間"""
        try:
            ntptime.settime()
            print("時間同步成功")
            return True
        except:
            print("時間同步失敗")
            return False

    def get_weather(self, location):
        """獲取天氣資料"""
        current_time = time.ticks_ms()

        # 檢查是否需要更新
        if self.weather_data and time.ticks_diff(current_time, 
self.last_update) < self.update_interval:
```

```
            return self.weather_data

        try:
            url = f"{self.base_url}?Authorization={self.api_key}
&locationName={location}"
            response = urequests.get(url)

            if response.status_code == 200:
                data = response.json()
                weather_element = data['records']['location'][0]
['weatherElement']

                # 解析天氣資料
                weather_info = {}
                for element in weather_element:
                    param_name = element['elementName']
                    if param_name == 'Wx':  # 天氣現象
                        weather_info['description'] = 
element['time'][0]['parameter']['parameterName']
                    elif param_name == 'PoP':  # 降雨機率
                        weather_info['rain_prob'] = 
element['time'][0]['parameter']['parameterName']
                    elif param_name == 'MinT':  # 最低溫度
                        weather_info['min_temp'] = element['time']
[0]['parameter']['parameterName']
                    elif param_name == 'MaxT':  # 最高溫度
                        weather_info['max_temp'] = element['time']
[0]['parameter']['parameterName']

                self.weather_data = weather_info
                self.last_update = current_time
                response.close()
                return weather_info
            else:
                response.close()
                return None
        except Exception as e:
            print(f"獲取天氣資料失敗: {e}")
            return None

class SnakeGame:
    def __init__(self, display, encoder):
        self.display = display
        self.encoder = encoder
        self.last_move = time.ticks_ms()

        # 方向變化限制
        self.last_direction_change = 0
        self.direction_change_cooldown = 100  # 縮短冷卻時間
```

```python
        self.pending_direction = None  # 儲存待處理的方向

        self.reset_game()

    def reset_game(self):
        """重置遊戲"""
        start_x = GRID_WIDTH // 2
        start_y = GRID_HEIGHT // 2
        self.snake = [(start_x, start_y), (start_x - 1, start_y)]
        self.direction = RIGHT
        self.pending_direction = None  # 重置待處理方向

        self.generate_food()

        self.game_over = False
        self.score = 0
        self.game_speed = 200

        self.old_snake = []
        self.old_food = None
        self._game_over_drawn = False

        # 清除編碼器的累積值
        self.encoder.clear_rotation()

    def generate_food(self):
        """生成食物位置"""
        while True:
            self.food = (random.randint(0, GRID_WIDTH - 1),
                         random.randint(0, GRID_HEIGHT - 1))
            if self.food not in self.snake:
                break

    def update_direction(self):
        """改進的方向更新邏輯"""
        if self.game_over:
            return

        current_time = time.ticks_ms()

        # 獲取旋轉值
        rotation = self.encoder.get_rotation()

        if rotation != 0:
            # 計算新方向
            if rotation > 0:
                # 順時針 = 右轉
                new_direction = (self.direction + 1) % 4
            else:
                # 逆時針 = 左轉
                new_direction = (self.direction - 1) % 4

            # 檢查是否可以改變方向
            opposite_directions = {UP: DOWN, DOWN: UP, LEFT: 
RIGHT, RIGHT: LEFT}

            # 如果新方向不是當前方向的相反方向
            if new_direction != 
opposite_directions[self.direction]:
                # 儲存為待處理方向，將在下次移動時應用
                self.pending_direction = new_direction
                direction_names = {UP: "上", RIGHT: "右", DOWN: 
"下", LEFT: "左"}
                print(f"方向預備改變: 
{direction_names[self.direction]} -> 
{direction_names[new_direction]}")

    def move_snake(self):
        """移動蛇 - 修正版本"""
        if self.game_over:
            return

        # 在移動前應用待處理的方向改變
        if self.pending_direction is not None:
            # 再次檢查是否為反向
            opposite_directions = {UP: DOWN, DOWN: UP, LEFT: 
RIGHT, RIGHT: LEFT}
            if self.pending_direction != 
opposite_directions[self.direction]:
                old_dir = self.direction
                self.direction = self.pending_direction
                direction_names = {UP: "上", RIGHT: "右", DOWN: 
"下", LEFT: "左"}
                print(f"方向已改變: {direction_names[old_dir]} -> 
{direction_names[self.direction]}")
            self.pending_direction = None

        head = self.snake[0]

        # 根據方向計算新頭部位置
        if self.direction == UP:
            new_head = (head[0], head[1] - 1)
        elif self.direction == DOWN:
            new_head = (head[0], head[1] + 1)
        elif self.direction == LEFT:
            new_head = (head[0] - 1, head[1])
        elif self.direction == RIGHT:
            new_head = (head[0] + 1, head[1])

        # 檢查碰撞
        if (new_head[0] < 0 or new_head[0] >= GRID_WIDTH or
            new_head[1] < 0 or new_head[1] >= GRID_HEIGHT):
            print("撞牆！")
            self.game_over = True
            return

        if new_head in self.snake[:-1]:
            print("撞到自己！")
            self.game_over = True
            return

        # 移動蛇
        self.snake.insert(0, new_head)

        # 檢查是否吃到食物
        if new_head == self.food:
            print("吃到食物!")
            self.score += 10
            self.generate_food()
            self.game_speed = max(80, self.game_speed - 3)
        else:
            self.snake.pop()

    def draw_boundary(self):
        """繪製遊戲邊界"""
        boundary_color = YELLOW

        game_left = OFFSET_X
        game_top = OFFSET_Y
        game_width = GRID_WIDTH * BLOCK_SIZE
        game_height = GRID_HEIGHT * BLOCK_SIZE

        self.display.rect(game_left - 2, game_top - 2,
                          game_width + 4, game_height + 4,
                          boundary_color)

        self.display.text(font, "<< Back", 5, 5, GRAY)

    def draw_block(self, x, y, color):
        """繪製一個方塊"""
        if 0 <= x < GRID_WIDTH and 0 <= y < GRID_HEIGHT:
            px = OFFSET_X + x * BLOCK_SIZE
            py = OFFSET_Y + y * BLOCK_SIZE
            self.display.fill_rect(px, py, BLOCK_SIZE, BLOCK_SIZE, 
color)
            if color != BLACK:
                self.display.rect(px, py, BLOCK_SIZE, BLOCK_SIZE, 
WHITE)

    def clear_block(self, x, y):
        """清除一個方塊"""
        if 0 <= x < GRID_WIDTH and 0 <= y < GRID_HEIGHT:
            px = OFFSET_X + x * BLOCK_SIZE
            py = OFFSET_Y + y * BLOCK_SIZE
            self.display.fill_rect(px, py, BLOCK_SIZE, BLOCK_SIZE, 
BLACK)

    def draw(self):
        """繪製遊戲畫面"""
        if self.game_over:
            if not self._game_over_drawn:
                self.display.fill(BLACK)
                center_x = SCREEN_WIDTH // 2
                center_y = SCREEN_HEIGHT // 2
                self.display.text(font, "GAME OVER", center_x - 
40, center_y - 30, WHITE)
                self.display.text(font, f"Score: {self.score}", 
center_x - 35, center_y - 10, WHITE)
                self.display.text(font, "Press button", center_x - 
45, center_y + 10, WHITE)
                self.display.text(font, "or << Back", center_x - 
40, center_y + 30, WHITE)
                self._game_over_drawn = True
        else:
            # 清除舊的蛇身
            for segment in self.old_snake:
                if segment not in self.snake:
                    self.clear_block(segment[0], segment[1])

            # 清除舊的食物
            if self.old_food and self.old_food != self.food:
                self.clear_block(self.old_food[0], 
self.old_food[1])

            # 繪製蛇
            for i, segment in enumerate(self.snake):
                if i == 0:
                    self.draw_block(segment[0], segment[1], BLUE)
                else:
                    self.draw_block(segment[0], segment[1], GREEN)

            # 繪製食物
            self.draw_block(self.food[0], self.food[1], RED)

            # 更新分數
            score_text = f"Score:{self.score}"
            score_x = OFFSET_X
            score_y = OFFSET_Y - 20
            self.display.fill_rect(score_x, score_y, 100, 16, 
BLACK)
            self.display.text(font, score_text, score_x, score_y, 
WHITE)

            # 儲存當前狀態
            self.old_snake = self.snake.copy()
            self.old_food = self.food

class SmartWatch:
    """智慧型手錶主類別"""
    def __init__(self, display, encoder):
        self.display = display
        self.encoder = encoder
        self.current_screen = 0  # 0: 主畫面, 1: 遊戲
        self.weather_api = WeatherAPI(CWA_API_KEY)
        self.rtc = RTC()
        self.weather_info = None
        self.wifi_connected = False

        # 初始化網路和時間
        self.init_network()

    def init_network(self):
        """初始化網路連接"""
        self.wifi_connected = 
self.weather_api.connect_wifi(WIFI_SSID, WIFI_PASSWORD)
        if self.wifi_connected:
            self.weather_api.sync_time()
            self.weather_info = 
self.weather_api.get_weather(LOCATION_NAME)

    def draw_clock_face(self):
        """繪製時鐘主畫面"""
        self.display.fill(BLACK)

        # 獲取當前時間
        year, month, day, weekday, hour, minute, second, _ = 
self.rtc.datetime()

        # 繪製時間 - 大字體
        time_str = f"{hour:02d}:{minute:02d}"
        self.display.text(font, time_str, 80, 80, WHITE)

        # 繪製日期
        date_str = f"{year}/{month:02d}/{day:02d}"
        self.display.text(font, date_str, 65, 110, GRAY)
        # 繪製星期
        weekdays = ["Mon", "Tue", "Wed", "Thu", "Fri", "Sat", 
"Sun"]
        weekday_str = weekdays[weekday]
        self.display.text(font, weekday_str, 100, 130, GRAY)

        # 繪製天氣資訊
        if self.weather_info:
            # 天氣描述
            weather_desc = self.weather_info.get('description', 
'N/A')
            # 簡化天氣描述以適應螢幕
            if len(weather_desc) > 8:
                weather_desc = weather_desc[:8]
            self.display.text(font, weather_desc, 70, 160, YELLOW)

            # 溫度範圍
            min_temp = self.weather_info.get('min_temp', 'N/A')
            max_temp = self.weather_info.get('max_temp', 'N/A')
            temp_str = f"{min_temp}~{max_temp}C"
            self.display.text(font, temp_str, 70, 180, ORANGE)

            # 降雨機率
            rain_prob = self.weather_info.get('rain_prob', 'N/A')
            rain_str = f"Rain:{rain_prob}%"
            self.display.text(font, rain_str, 70, 200, BLUE)
        else:
            self.display.text(font, "No Weather", 70, 160, GRAY)

        # 繪製導航提示
        self.display.text(font, ">>", 210, 110, WHITE)
        self.display.text(font, "Game", 195, 130, GRAY)

    def update_weather(self):
        """更新天氣資料"""
        if self.wifi_connected:
            new_weather = 
self.weather_api.get_weather(LOCATION_NAME)
            if new_weather:
                self.weather_info = new_weather

    def run(self):
        """主執行迴圈 - 修正版本"""
        last_update = time.ticks_ms()
        last_minute = -1
        snake_game = None

        # 初始化主畫面
        self.draw_clock_face()

        while True:
            current_time = time.ticks_ms()

            if self.current_screen == 0:  # 主畫面
                # 使用 peek 來檢查是否有旋轉，避免消耗數據
                if self.encoder.peek_rotation() > 0:
                    # 確認要進入遊戲，才真正讀取值
                    rotation = self.encoder.get_rotation()
                    if rotation > 0:
                        print("進入遊戲模式")
                        self.current_screen = 1
                        if snake_game is None:
                            snake_game = SnakeGame(self.display, 
self.encoder)
                        else:
                            snake_game.reset_game()
                        self.display.fill(BLACK)
                        snake_game.draw_boundary()
                        snake_game.draw()

                # 更新時間顯示
                _, _, _, _, _, minute, _, _ = self.rtc.datetime()
                if minute != last_minute:
                    self.draw_clock_face()
                    last_minute = minute

                # 更新天氣
                if time.ticks_diff(current_time, last_update) > 
1800000:
                    self.update_weather()
                    last_update = current_time

            elif self.current_screen == 1:  # 遊戲畫面
                if snake_game:
                    if snake_game.game_over:
                        # 遊戲結束狀態
                        if self.encoder.is_button_pressed():
                            print("重新開始遊戲")
                            snake_game.reset_game()
                            self.display.fill(BLACK)
                            snake_game.draw_boundary()
                            snake_game.draw()
                        elif self.encoder.peek_rotation() < 0:
                            rotation = self.encoder.get_rotation()
                            if rotation < 0:
                                print("返回主畫面")
                                self.current_screen = 0
                                self.draw_clock_face()
                    else:
                        # 遊戲進行中 - 只更新方向，不讀取旋轉
                        snake_game.update_direction()
```

```
                        # 定時移動蛇
                        if time.ticks_diff(current_time, 
snake_game.last_move) >= snake_game.game_speed:
                            snake_game.move_snake()
                            snake_game.last_move = current_time
                            snake_game.draw()
```

```
            time.sleep_ms(10)
```

測試過程與結果

測試過程還算順利，只有遇到一點點的BUG，例如編碼器靈敏度太高，或者是編碼器彈跳，以及顯示范圍跟廠商說的不太一樣，但最終仍然修正了問題。

遇到的問題與解法

- 解碼器設定：設定的感覺不好抓，原本是一格一格控制的，這會導致不好操作，現在則是改為轉約45度為控制，這樣控制起來比較直覺。
- 顯示區域:圓形的顯示器並不好控制顯示范圍，現在只能先縮小顯示范圍，未來希望能改成圓形的顯示范圍
- 硬體資源限制：ESP32 的記憶體和處理能力有限，需要捨棄臃腫的功能，例如中文字庫，不然很容易爆ROM

結論與心得

專案成果

本專案成功實現了一個結合日常實用功能與娛樂性的嵌入式系統應用。透過 ESP32 搭配 GC9A01 圓形顯示器和旋轉編碼器，打造出一個迷你桌面裝飾的原型。系統整合了以下主要功能：

1. **時間顯示功能**：即時顯示當前時間、日期和星期，發揮桌面裝飾的基本功能
2. **天氣狀況顯示**：透過中央氣象局 Open Data API 獲取即時天氣資料
3. **貪吃蛇遊戲**：經典遊戲的嵌入式實現，提供娛樂功能，成功降低使用者的工作效率
4. **直覺操作**：使用旋轉編碼器實現簡潔的單手操作，感覺很適合改成桌面的智慧旋鈕

未來展望

此專案具有很大的擴展潛力：

1. **功能擴充**：可加入更多實用功能，如音樂播放控制、久坐提醒、遊戲每日獎勵通知等
2. **加強互動性**：加裝揚聲器、震動馬達，讓裝置可以使勁全力干擾用戶
3. **外觀設計**：設計 3D 列印外殼，打造真正可用的桌面小廢物
4. **更新UI**：改良UI使UI成為動態UI，方便看著發呆
5. **連接性增強**：加入藍牙功能，與手機 App 連動，方便吸引用戶分心