Temat: „Implementacja klasycznej gry Pong na ekranie LCD płytki Nucleo L476RG”

1. Temat Projektu

Klasyczna gry Pong na ekranie LCD płytki Nucleo L476RG w środowisku STM32CubeIDE, wzbogacona o sterowanie za pośrednictwem interfejsu UART oraz inteligentny algorytm śledzenia trajektorii piłki, umożliwiający automatyczne pozycjonowanie paletki w celu skutecznego odbicia. Projekt łączy w sobie aspekty grafiki wbudowanej, komunikacji szeregowej oraz algorytmiki predykcyjnej, zapewniając dynamiczną i responsywną rozgrywkę.

2. Wstęp teoretyczny

Implementacja została zrealizowana w środowisku STM32CubeIDE dla mikrokontrolera STM32L476RG, wykorzystując wyświetlacz OLED SSD1306 sterowany poprzez magistralę I2C oraz komunikację UART do interakcji użytkownika. Program rozpoczyna działanie od inicjalizacji niezbędnych peryferiów w funkcji main() oraz struktur jako zmiennych globalnych, gdzie wywoływane są procedury konfigurujące wyprowadzenia mikrokontrolera, magistralę I2C oraz interfejs UART2 umożliwiający sterowanie paletką gracza. Po inicjalizacji wyświetlacza definiowane są początkowe pozycje paletek, a interfejs UART przechodzi w tryb asynchronicznego odbioru danych.

Obsługa grafiki bazuje na funkcji Update_Screen(), która w każdej iteracji odświeża wyświetlacz, usuwając poprzednie klatki i rysując aktualne pozycje elementów gry. Wykorzystano tutaj funkcje biblioteki SSD1306, umożliwiające dynamiczną wizualizację obiektów takich jak paletki i piłka. Końcowe wywołanie ssd1306_UpdateScreen() przesyła wygenerowaną klatkę do wyświetlacza, zapewniając płynność animacji.

Ruch piłki jest realizowany za pośrednictwem funkcji updateSquarePosition(), która modyfikuje współrzędne x i y na podstawie wartości dx i dy. Odbicia od ścian oraz kolizje z paletkami wykrywane są poprzez analizę warunków logicznych. Odbicie od górnej i dolnej krawędzi ekranu powoduje inwersję dy, a kolizja z paletką gracza zmienia znak dx, symulując odbicie. Podobna zasada dotyczy paletki komputera, której ruch jest dodatkowo kontrolowany przez algorytm przewidujący trajektorię piłki. Jeśli piłka przekroczy lewą krawędź, licznik punktów licznik_punktow zostaje zwiększony, a wynik wysyłany przez UART do użytkownika.

Sterowanie paletką gracza odbywa się za pośrednictwem interfejsu UART, który odbiera pojedyncze znaki (w i s - z klawiatury) i odpowiednio koryguje współrzędne gracz.y_pos. Obsługa wejścia realizowana jest w funkcji receive_uart_command(), która analizuje otrzymane dane i dostosowuje pozycję paletki w górę lub w dół. Działanie w trybie asynchronicznym zapewnia

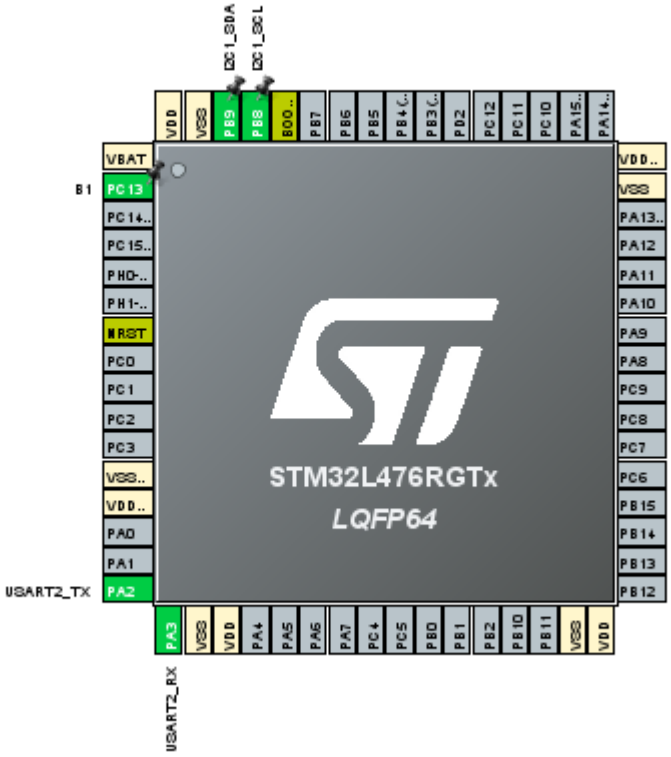
natychmiastową reakcję na polecenia bez zakłócania płynności gry. Funkcja HAL_UART_RxCpltCallback() obsługuje odbiór danych w trybie przerwań, umożliwiając bieżące przetwarzanie wejściowych komend.

Przeciwnik wykorzystuje funkcję Estimate_Pos(), analizując trajektorię piłki. Jeśli piłka zmierza w stronę paletki komputera, algorytm oblicza przewidywaną współrzędną y w momencie kontaktu, wykorzystując zależność pomiędzy czasem dotarcia a ruchem piłki. Obliczona wartość jest ograniczana do wymiarów ekranu, a następnie wykorzystywana do dostosowania komputer.y_pos, co pozwala na precyzyjne ustawienie paletki w optymalnej pozycji.

Gra może zostać zresetowana poprzez przycisk B1, co wywołuje HAL_GPIO_EXTI_Callback(). Po wykryciu zdarzenia następuje wywołanie SetDefault(), resetujące wszystkie parametry gry do wartości początkowych.

Implementacja bazuje na modularnej strukturze, gdzie każda funkcjonalność realizowana jest przez dedykowaną funkcję. Asynchroniczna komunikacja przez UART, dynamiczna aktualizacja grafiki i algorytm predykcji ruchu piłki tworzą płynną i angażującą rozgrywkę. Możliwe usprawnienia obejmują dodanie zaawansowanego systemu punktacji, efektów dźwiękowych oraz trybu wieloosobowego.

3. Konfiguracja pinów



PC_13 (B1) - pin odpowiadający za niebieski przycisk na płytce

PB8, PB9 – piny odpowiadające za połączenie płytki z ekranem

PA2, PA3 – piny odpowiadające za komunikacje z interfejsem uart

4. Analiza poszczególnych części kodu

Inicjalizacja bibliotek

```
#include "main.h"
#include "ssd1306.h"
#include "ssd1306_tests.h"
#include "ssd1306_fonts.h"
#include <stdbool.h>
#include <stdio.h>
#include <string.h>
```

Zdefiniowanie wymiarów paletki

```
#define RECHEIGHT 10
#define RECWIDTH 4
```

Deklaracja obsługi interfejsów

I2C_HandleTypeDef hi2c1; **Interfejs do sterowania wyświetlaczem OLED SSD1306**

UART_HandleTypeDef huart2; **Interfejs do obsługi komunikacji UART**

Zmienne sterujące grą

int licznik_punktow = 0; **Licznik punktów gracza**

char point_buffer[50]; **Bufor do wysyłania wyników przez UART**

int estimated_pos; **Przewidywana wartość na osi y paletki komputera**

volatile bool newInput = false; **Flaga oznaczająca nowy znak z UART**

volatile char inputChar = '\0'; **Przechowywany znak wejściowy**

Inicjalizacja prototypów funkcji

void SystemClock_Config(void);

static void MX_GPIO_Init(void);

static void MX_I2C1_Init(void);

static void MX_USART2_UART_Init(void);

Deklaracji pozycji początkowych piłki i wartości wektora zmiany

static int x = (SSD1306_WIDTH / 2) -10;

static int y = SSD1306_HEIGHT / 2;

int dx = 2;

int dy = 2;

Definicja struktury paletek z 2 zmiennymi jako współrzędne lewego dolnego rogu paletki

```
struct paletka {
    int x_pos;
    int y_pos;
};
```

Deklaracja paletek

struct paletka gracz;

struct paletka komputer;

Funkcja ustawiająca domyślne wartości

void SetDefault() {

licznik_punktow=0; **ustawienie punktów na 0**

```
    dx = 2; wektor zmiany w osi x
    dy = 2; wektor zmiany w osi y
    x = SSD1306_WIDTH / 2; pozycja x na środku ekranu
    y = SSD1306_HEIGHT / 2; pozycja y na środku ekranu
}
```

Analogiczna funkcja obsługującą przegraną

```
void Lose()
{
    dx=2;
    dy=2;
    x=SSD1306_WIDTH / 2;
    y=SSD1306_HEIGHT / 2;
}
```

Obsługa za pomocą przerwań wciśnięcia przycisku

```
void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)
{
     if (GPIO_Pin == B1_Pin)
     {
          SetDefault(); restart gry
     }
}
```

Funkcja aktualizująca ekran OLED

```
void Update_Screen() {
    ssd1306_Fill(Black); Wypełnienie ekranu kolorem czarnym
```

ssd1306_DrawRectangle(gracz.x_pos, gracz.y_pos, gracz.x_pos + RECWIDTH, gracz.y_pos + RECHEIGHT, White); **Rysowanie paletki gracza**

ssd1306_DrawRectangle(komputer.x_pos, komputer.y_pos, komputer.x_pos + RECWIDTH, komputer.y_pos + RECHEIGHT, White); **Rysowanie paletki komputera**

ssd1306_DrawRectangle(x, y, x + 5, y + 5, White); **Rysowanie piłki**

ssd1306_UpdateScreen(); **Aktualizacja ekranu**

}

Funkcja aktualizująca pozycję piłki i wykrywająca kolizje

void updateSquarePosition() {

x += dx;

y += dy;

} **Zmiana zmiennych x i y odpowiadających pozycji piłki o wartość wektorową**

Odbicie od paletki gracza

if (x < gracz.x_pos + RECWIDTH && x + 5 > gracz.x_pos && y < gracz.y_pos + RECHEIGHT && y + 5 > gracz.y_pos) {

x = gracz.x_pos + RECWIDTH;

dx = -dx; **Zmiana kierunku piłki**

} **warunek sprawdzający czy piłka dotyka paletki gracza**

Odbicie od paletki komputera

if (x + 5 > komputer.x_pos && x < komputer.x_pos + RECWIDTH && y < komputer.y_pos + RECHEIGHT && y + 5 > komputer.y_pos) {

x = komputer.x_pos - 5;

dx = -dx;

} **warunek sprawdzający czy piłka dotyka paletki komputera**

Przekroczenie lewej krawędzi ekranu – gracz traci punkt

if (x < 3 && dx < 0) {

licznik_punktow++; **zwiększenie wartości punktów o 1**

sprintf(point_buffer, "Ilosc punktow: %d", licznik_punktow); **skonwertowanie do typu string napisu pokazującego ilość punktów**

HAL_UART_Transmit(&point_buffer, (uint8_t *)point_buffer, strlen(point_buffer), HAL_MAX_DELAY); **wypisanie przez interfejs uart liczby punktów**

Lose(); **obsługa przegranej**

}

Odbicie od prawej ściany

```
if (x > SSD1306_WIDTH - 7 && dx > 0) {
    x = SSD1306_WIDTH - 7;
    dx = -dx;
}
```

Odbicie od górnej i dolnej ściany

```
if (y < 0 && dy < 0) {
    y = 0;
    dy = -dy;
}
if (y > SSD1306_HEIGHT - 7 && dy > 0) {
    y = SSD1306_HEIGHT - 7;
    dy = -dy;
}
}
```

Funkcja odbierająca polecenia sterowania przez UART

```
void receive_uart_command() {
    char direction = __io_getchar(); odczytanie kierunku
    switch(direction) {
        case 'w': Ruch w górę
            if (gracz.y_pos > 0) gracz.y_pos--;
            break;
```

```
        case 's':  Ruch w dół
            if (gracz.y_pos < SSD1306_HEIGHT - RECHEIGHT) gracz.y_pos++;
            break;
    }
}
```

Funkcja przewidująca pozycję piłki i dostosowująca ruch paletki komputera

```
void Estimate_Pos() {
    if (dx > 0) {
        int distance_x = komputer.x_pos - x; Piłka porusza się w stronę komputera

        if (distance_x > 0) {
            int time_to_reach = distance_x / dx;
            int estimated_y = y + dy * time_to_reach;
                        Ograniczenie estimated_y do obszaru ekranu
            if (estimated_y < 0) {
                estimated_y = 0;
            } else if (estimated_y > SSD1306_HEIGHT - RECHEIGHT) {
                estimated_y = SSD1306_HEIGHT - RECHEIGHT;
            }

            estimated_pos = estimated_y;
        }
    }

                Ruch paletki przeciwnika w kierunku przewidywanej pozycji
    if (komputer.y_pos < estimated_pos && komputer.y_pos < SSD1306_HEIGHT -
RECHEIGHT) {
        komputer.y_pos++;
```

```
    } else if (komputer.y_pos > estimated_pos && komputer.y_pos > 0) {
        komputer.y_pos--;
    }
}
```

Pętla while

updateSquarePosition(); **aktualizacja pozycji piłki**
receive_uart_command(); **otrzymanie z interfejsu uart inputu z klawiatury**
Estimate_Pos(); **obliczanie pozycji**
Update_Screen(); **aktualizacja ekranu**
HAL_Delay(10); **odzczekanie 10 ms dla płynności**