ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH

TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN

KHOA KỸ THUẬT MÁY TÍNH

----------ଓ★ଌ----------

BÁO CÁO MINI PROJECT
MÔN THIẾT KẾ HỆ THỐNG NHÚNG

Lớp: CE224.O12.1

Giảng viên hướng dẫn: ThS. Chung Quang Khánh

Nhóm sinh viên thực hiện:

1. Đào Phước Tài (50%) 21521391
2. Nguyễn Anh Khôi (50%) 21520301

TP.Hồ Chí Minh, ngày 21 tháng 12 năm 2023

I. MÔ TẢ GAME

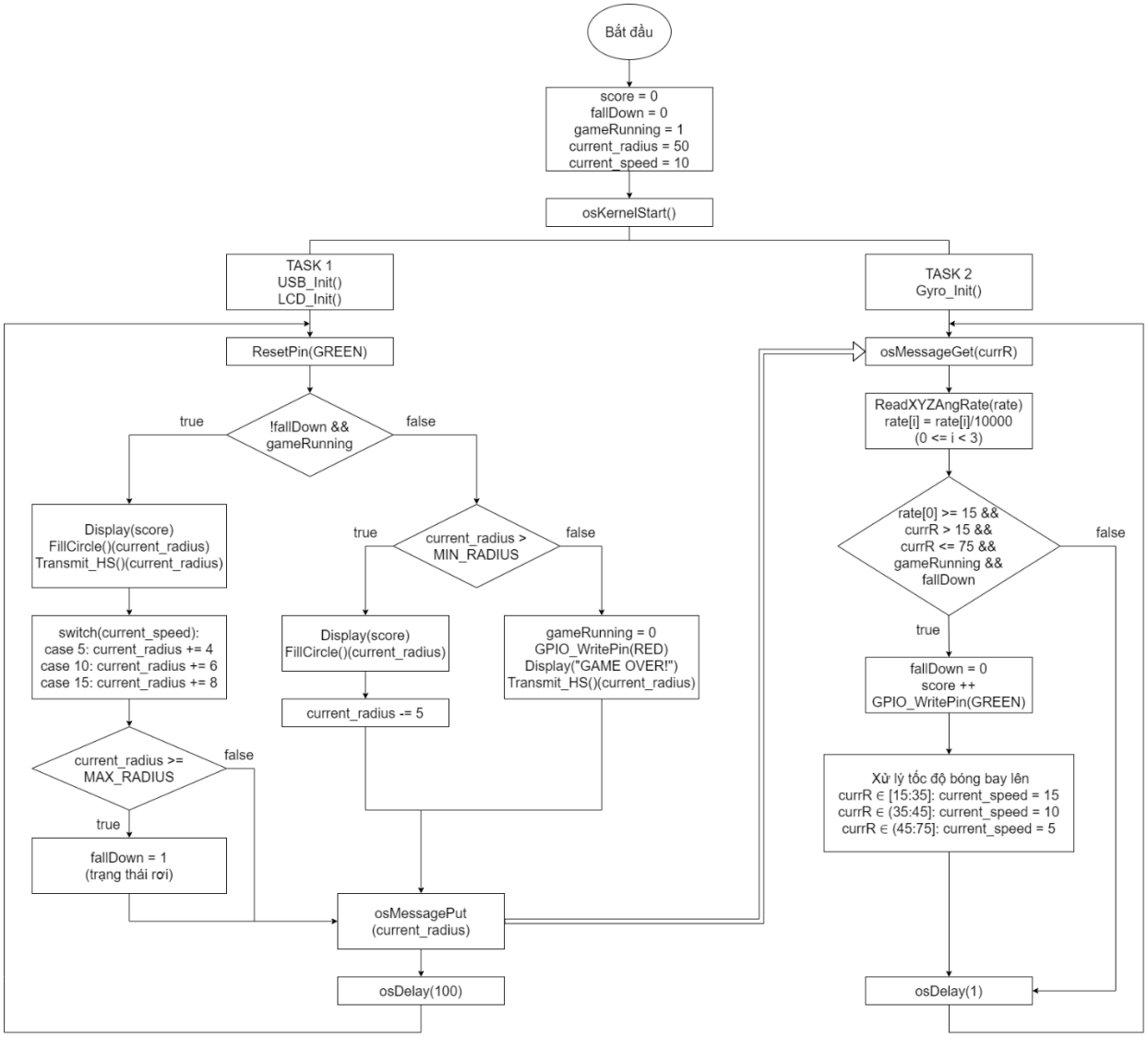
1. Nguyên lý
   - Đầu tiên, hệ thống ở trạng thái cân bằng khởi đầu.
   - Người chơi sẽ hạ và nâng KIT để thực hiện động tác tâng bóng, khi đó trên màn hình LCD sẽ mô phỏng trạng thái trái bóng được nâng lên và rơi xuống.
   - Mục tiêu của người chơi là phải nâng KIT đúng lúc trái bóng rơi xuống, đập vào "mặt vợt" và tâng lên lại, người chơi được tính điểm, đèn xanh chớp 1 lần.
   - Nếu người chơi nâng đúng, tùy thuộc vào mức độ chính xác mà bóng sẽ nảy lên với tốc độ khác nhau.
   - Nếu người tâng "hụt" thì bóng sẽ rớt và GAME OVER, đèn đỏ được bật và giữ cho tới khi reset.
2. Yêu cầu

Yêu cầu	Điểm
Hiển thị được giao diện ban đầu	2 điểm
Hiện thực được thao tác tâng bóng tại chỗ (điểm giữa màn hình)	3 điểm
Xuất thông số độ cao khi bóng được tâng ra máy tính qua Virtual Com Port	2 điểm
Hiện thực được chức năng tính điểm	3 điểm
Hiện thực được việc bóng bay theo 1 chiều dựa trên phương và chiều của lúc nâng bóng (nếu bóng bay ra rìa LCD sẽ đập ngược trở lại)	1 điểm
Hiện thực được việc bóng bay theo 2 chiều dựa trên phương và chiều của lúc nâng bóng (nếu bóng bay ra rìa LCD sẽ đập ngược trở lại)	1 điểm

II. BÀI TẬP

1. Bài tập 1: Thiết kế mô hình phần cứng cứng thiết cho yêu cầu trên: cần dùng những phần cứng nào, sử dụng các giao thức giao tiếp gì giữa các phần cứng? (15% số điểm).
   - Các thành phần phần cứng cần thiết cho thiết kế này bao gồm:
     - Kit STM32F4 Discovery.
     - Màn hình hiển thị LCD.
     - Cảm biến (gyroscope) và nút nhấn (reset).
     - Cáp kết nối USB.
   - Các giao thức giao tiếp giữa các phần cứng trên: SPI, I2C, UART.
2. Bài tập 2: Sử dụng RTOS, thiết mô hình phần mềm cho yêu cầu trên: nêu công việc của từng task, luồng xử lý dữ liệu như thế nào? (15% số điểm).
   - Phần mềm trên được chia làm 2 task:
     - Task01 có nhiệm vụ xử lý hiển thị LCD, truyền dữ liệu độ cao qua máy tính thông qua USB và xử lý các logic của trò chơi.
     - Task02 có nhiệm vụ lấy dữ liệu gyroscope, lấy dữ liệu độ cao từ Task01 và xử lý logic việc tính điểm, tốc độ tăng bóng.
     - Task01 và Task02 giao tiếp với nhau qua Message Queue: Task01 chuyển thông số độ cao qua phương thức osMessagePut(), Task02 lấy thông số này thông qua osMessageGet() để phục vụ các nhiệm vụ xử lý.

3. <u>Bài tập 3:</u> Thiết kế lưu đồ giải thuật xử lý cho yêu cầu trên? (30% số điểm).



4. Bài tập 4: Hiện thực hệ thống và báo cáo kết quả? (40% số điểm).
   - Code hệ thống

```
#include "main.h"
#include "cmsis_os.h"
#include "usb_device.h"

/* Private includes ----------------------------------------------------
-------*/
/* USER CODE BEGIN Includes */
#include "stdio.h"
#include "stm32f429i_discovery_lcd.h"
#include "string.h"
#include "usbd_cdc_if.h"
#include "stm32f429i_discovery_gyroscope.h"
/* USER CODE END Includes */

/* Private variables ---------------------------------------------------
-------*/
osThreadId TaskDisplayLCDHandle;
osThreadId TaskGyroScopeHandle;
osMessageQId myQueue01Handle;
/* USER CODE BEGIN PV */
static int current_radius = 50;
static int current_speed = 10;
static int fallDown = 0;
static int gameRunning = 1;
static int score = 0;

#define MAX_RADIUS_OF_HIGH_SPEED       120
#define MAX_RADIUS_OF_MEDIUM_SPEED            110
#define MAX_RADIUS_OF_LOW_SPEED               100
#define MIN_RADIUS        15

#define HIGH_SPEED        15
#define MEDIUM_SPEED      10
#define LOW_SPEED         5
/* USER CODE END PV */

/* Private function prototypes -----------------------------------------
-------*/
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
void StartTask01(void const * argument);
void StartTask02(void const * argument);

/* USER CODE BEGIN PFP */

/* USER CODE END PFP */

/* Private user code ---------------------------------------------------
-------*/
/* USER CODE BEGIN 0 */
```

```
/* USER CODE END 0 */

/**
  * @brief  The application entry point.
  * @retval int
  */
int main(void)
{
  /* USER CODE BEGIN 1 */

  /* USER CODE END 1 */

  /* MCU Configuration--------------------------------------------------------*/

  /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
  HAL_Init();

  /* USER CODE BEGIN Init */

  /* USER CODE END Init */

  /* Configure the system clock */
  SystemClock_Config();

  /* USER CODE BEGIN SysInit */

  /* USER CODE END SysInit */

  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  /* USER CODE BEGIN 2 */

  /* USER CODE END 2 */

  /* USER CODE BEGIN RTOS_MUTEX */
  /* add mutexes, ... */
  /* USER CODE END RTOS_MUTEX */

  /* USER CODE BEGIN RTOS_SEMAPHORES */
  /* add semaphores, ... */
  /* USER CODE END RTOS_SEMAPHORES */

  /* USER CODE BEGIN RTOS_TIMERS */
  /* start timers, add new ones, ... */
  /* USER CODE END RTOS_TIMERS */

  /* Create the queue(s) */
  /* definition and creation of myQueue01 */
  osMessageQDef(myQueue01, 16, uint8_t);
  myQueue01Handle = osMessageCreate(osMessageQ(myQueue01), NULL);

  /* USER CODE BEGIN RTOS_QUEUES */
  /* add queues, ... */
```

```
  /* USER CODE END RTOS_QUEUES */

  /* Create the thread(s) */
  /* definition and creation of TaskDisplayLCD */
  osThreadDef(TaskDisplayLCD, StartTask01, osPriorityNormal, 0, 256);
  TaskDisplayLCDHandle = osThreadCreate(osThread(TaskDisplayLCD),
NULL);

  /* definition and creation of TaskGyroScope */
  osThreadDef(TaskGyroScope, StartTask02, osPriorityNormal, 0, 256);
  TaskGyroScopeHandle = osThreadCreate(osThread(TaskGyroScope), NULL);

  /* USER CODE BEGIN RTOS_THREADS */
  /* add threads, ... */
  /* USER CODE END RTOS_THREADS */

  /* Start scheduler */
  osKernelStart();

  /* We should never get here as control is now taken by the scheduler
*/
  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {
    /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */
  }
  /* USER CODE END 3 */
}

/**
  * @brief System Clock Configuration
  * @retval None
  */
void SystemClock_Config(void)
{
  RCC_OscInitTypeDef RCC_OscInitStruct = {0};
  RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};

  /** Configure the main internal regulator output voltage
  */
  __HAL_RCC_PWR_CLK_ENABLE();
  __HAL_PWR_VOLTAGESCALING_CONFIG(PWR_REGULATOR_VOLTAGE_SCALE1);

  /** Initializes the RCC Oscillators according to the specified
parameters
  * in the RCC_OscInitTypeDef structure.
  */
  RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSE;
  RCC_OscInitStruct.HSEState = RCC_HSE_ON;
  RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
  RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSE;
```

```
  RCC_OscInitStruct.PLL.PLLM = 8;
  RCC_OscInitStruct.PLL.PLLN = 336;
  RCC_OscInitStruct.PLL.PLLP = RCC_PLLP_DIV2;
  RCC_OscInitStruct.PLL.PLLQ = 7;
  if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
  {
    Error_Handler();
  }

  /** Initializes the CPU, AHB and APB buses clocks
  */
  RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
                              |RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
  RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
  RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
  RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV4;
  RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV4;

  if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_5) !=
HAL_OK)
  {
    Error_Handler();
  }
}

/**
  * @brief GPIO Initialization Function
  * @param None
  * @retval None
  */
static void MX_GPIO_Init(void)
{
  GPIO_InitTypeDef GPIO_InitStruct = {0};
/* USER CODE BEGIN MX_GPIO_Init_1 */
/* USER CODE END MX_GPIO_Init_1 */

  /* GPIO Ports Clock Enable */
  __HAL_RCC_GPIOH_CLK_ENABLE();
  __HAL_RCC_GPIOB_CLK_ENABLE();
  __HAL_RCC_GPIOA_CLK_ENABLE();
  __HAL_RCC_GPIOG_CLK_ENABLE();

  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(GPIOG, GPIO_PIN_13|GPIO_PIN_14, GPIO_PIN_RESET);

  /*Configure GPIO pins : PG13 PG14 */
  GPIO_InitStruct.Pin = GPIO_PIN_13|GPIO_PIN_14;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_HIGH;
  HAL_GPIO_Init(GPIOG, &GPIO_InitStruct);

/* USER CODE BEGIN MX_GPIO_Init_2 */
/* USER CODE END MX_GPIO_Init_2 */
}
```

```
/* USER CODE BEGIN 4 */

/* USER CODE END 4 */

/* USER CODE BEGIN Header_StartTask01 */
/**
  * @brief  Function implementing the TaskDisplayLCD thread.
  * @param  argument: Not used
  * @retval None
  */
/* USER CODE END Header_StartTask01 */
void StartTask01(void const * argument)
{
  /* init code for USB_DEVICE */
  MX_USB_DEVICE_Init();
  /* USER CODE BEGIN 5 */
  BSP_LCD_Init();
  BSP_LCD_LayerDefaultInit(1, SDRAM_DEVICE_ADDR);
  BSP_LCD_SelectLayer(1);
  BSP_LCD_DisplayOn();
  BSP_LCD_SetBackColor(LCD_COLOR_GREEN);
  BSP_LCD_SetTextColor(LCD_COLOR_BLACK);
  BSP_LCD_Clear(LCD_COLOR_GREEN);
  float x angRate;
  char str_score [50];
  char str_currentHigh [50];
  /* Infinite loop */
  for(;;)
  {
        sprintf(str_score,"Score: %d", score);
        sprintf(str_currentHigh, "Current high: %d\n", current_radius);
        BSP_LCD_Clear(LCD_COLOR_GREEN);
        HAL_GPIO_WritePin(GPIOG, GPIO_PIN_13, GPIO_PIN_RESET);

        if(!fallDown && gameRunning){
              CDC_Transmit_HS(str_currentHigh,
strlen(str_currentHigh));
              BSP_LCD_DisplayStringAtLine(1,str_score);
              BSP_LCD_FillCircle(120, 160, current_radius);
              switch (current_speed) {
                   case HIGH_SPEED:{
                         current_radius += 8;
                         if(current_radius >=
MAX_RADIUS_OF_HIGH_SPEED)
                               fallDown = 1;
                         break;
                   }
                   case MEDIUM_SPEED:{
                         current_radius += 6;
                         if(current_radius >=
MAX_RADIUS_OF_MEDIUM_SPEED)
                               fallDown = 1;
                         break;
                   }
```

```
                    case LOW_SPEED:{
                          current_radius += 4;
                          if(current_radius >= MAX_RADIUS_OF_LOW_SPEED)
                                fallDown = 1;
                          break;
                    }
                    default:
                          break;
              }
        }
        else{
              if(current_radius > MIN_RADIUS){
                    BSP_LCD_DisplayStringAtLine(1,str_score);
                    BSP_LCD_FillCircle(120, 160, current_radius);
                    current_radius -= 5;
              }
              else{
                    gameRunning = 0;
                    BSP_LCD_Clear(LCD_COLOR_GREEN);
                    BSP_LCD_DisplayStringAtLine(5, "  GAME OVER!");
                    HAL_GPIO_WritePin(GPIOG, GPIO_PIN_14,
GPIO_PIN_SET);
                    HAL_GPIO_WritePin(GPIOG, GPIO_PIN_13,
GPIO_PIN_RESET);
                    CDC_Transmit_HS(str_currentHigh,
strlen(str_currentHigh));
              }
        }
        osMessagePut(myQueue01Handle, current_radius, osWaitForever);
        osDelay(100);
  }
  /* USER CODE END 5 */
}

/* USER CODE BEGIN Header_StartTask02 */
/**
* @brief Function implementing the TaskGyroScope thread.
* @param argument: Not used
* @retval None
*/
/* USER CODE END Header_StartTask02 */
void StartTask02(void const * argument)
{
  /* USER CODE BEGIN StartTask02 */
       BSP_GYRO_Init();
       float L3GD20_AngRate[3];
  /* Infinite loop */
  for(;;){
       osEvent event = osMessageGet(myQueue01Handle, osWaitForever);
       int currentRadiusFromMessQ = event.value.v;

       L3GD20_ReadXYZAngRate(L3GD20_AngRate);
       for(int i = 0; i < 3; i++){
             L3GD20_AngRate[i] = L3GD20_AngRate[i]/10000;
       }
```

```
      if(L3GD20_AngRate[0] >= 15 && currentRadiusFromMessQ > 15 &&
currentRadiusFromMessQ <= 75 && gameRunning && fallDown){
            fallDown = 0;
            score++;
            HAL_GPIO_WritePin(GPIOG, GPIO_PIN_13, GPIO_PIN_SET);
            if(currentRadiusFromMessQ <= 35)
                  current_speed = HIGH_SPEED;
            else if (currentRadiusFromMessQ > 35 &&
currentRadiusFromMessQ <= 45)
                  current_speed = MEDIUM_SPEED;
            else
                  current_speed = LOW_SPEED;
      }
    osDelay(1);
  }
  /* USER CODE END StartTask02 */
}

/**
  * @brief  Period elapsed callback in non blocking mode
  * @note   This function is called  when TIM6 interrupt took place,
inside
  * HAL_TIM_IRQHandler(). It makes a direct call to HAL_IncTick() to
increment
  * a global variable "uwTick" used as application time base.
  * @param  htim : TIM handle
  * @retval None
  */
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
{
  /* USER CODE BEGIN Callback 0 */

  /* USER CODE END Callback 0 */
  if (htim->Instance == TIM6) {
    HAL_IncTick();
  }
  /* USER CODE BEGIN Callback 1 */

  /* USER CODE END Callback 1 */
}

/**
  * @brief  This function is executed in case of error occurrence.
  * @retval None
  */
void Error_Handler(void)
{
  /* USER CODE BEGIN Error_Handler_Debug */
  /* User can add his own implementation to report the HAL error return
state */
  __disable_irq();
  while (1)
  {
  }
  /* USER CODE END Error_Handler_Debug */
```

```
}

#ifdef  USE_FULL_ASSERT
/**
  * @brief  Reports the name of the source file and the source line
number
  *         where the assert_param error has occurred.
  * @param  file: pointer to the source file name
  * @param  line: assert_param error line source number
  * @retval None
  */
void assert_failed(uint8_t *file, uint32_t line)
{
  /* USER CODE BEGIN 6 */
  /* User can add his own implementation to report the file name and
line number,
     ex: printf("Wrong parameters value: file %s on line %d\r\n", file,
line) */
  /* USER CODE END 6 */
}
#endif /* USE_FULL_ASSERT */
```

- Link báo cáo kết quả: *https://youtu.be/WsZNiXzZZtw*