Druga domača naloga OR

Rok Švikart

Contents

Uvod 3
Pomožne funkcije 3
Glavna zanka 5
Celotna koda 7

Uvod

Za nalogo sem se odločil, da uporabim zaslon na dotik. Naredil sem preprosto igro v programskem jeziku C kjer pokamo krogce. Za osnovo sem uporabil projekt STM32H750B-DK_BSP_C_Basic.

Pomožne funkcije

Funkcija Clear_LCD() počisti zaslon. Funkcija Get_Circle() počisti zaslon in izriše krog na naključni lokaciji na zaslonu. Funkcija Explode_Circle() simulira pok kroga.

```
static void Clear_LCD()
{
    /* Clear the LCD */
      UTIL_LCD_SetBackColor(UTIL_LCD_COLOR_BLACK);
      UTIL_LCD_Clear(UTIL_LCD_COLOR_BLACK);
      BSP_LCD_FillRect(0, 0, 0, x_size, y_size, UTIL_LCD_COLOR_BLACK);
}

void Get_Circle(int32_t *XPos,int32_t *YPos, int32_t rad)
{
    Clear_LCD();
    *XPos = (rand()%(x_size - 2*rad)) + rad;
    *YPos = (rand()%(y_size - 2*rad)) + rad;

    UTIL_LCD_FillCircle(*XPos,*YPos,rad,UTIL_LCD_COLOR_RED);
}

void Explode_Circle(int32_t *XPos,int32_t *YPos, int32_t rad)
{
    Clear_LCD();
    UTIL_LCD_DrawCircle(*XPos,*YPos,rad + 10,UTIL_LCD_COLOR_RED);

    HAL_Delay(100);

    Get_Circle(XPos,YPos, rad);

}
```

Pomožni funkciji DisplayStartScreen() in DisplayEndScreen() prikažeta začetni in končni zaslon.

Funkcija HAL_TIM_PeriodElapsedCallback in MX_TIM16_Init sta funkciji za delovanje časovnika, kjer v prvi definiramo kaj naj se zgodi ko preteče nastavljen čas. Medtem ko se druga uporabi za inicializacijo. V programu je prekinitev uporabljena, da konča igro in prikaže končni zaslon.

```
static void MX_TIM16_Init(void)
{
    HAL_NVIC_SetPriority(TIM16_IRQn, TICK_INT_PRIORITY, 0U);
    HAL_NVIC_EnableIRQ(TIM16_IRQn);
    __HAL_RCC_TIM16_CLK_ENABLE();

  htim16.Instance = TIM16;
  htim16.Init.Prescaler = 1000000;
  htim16.Init.CounterMode = TIM_COUNTERMODE_UP;
  // gets set in the game
//  htim16.Init.Period = 30000;
  htim16.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
  htim16.Init.RepetitionCounter = 0;
  htim16.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
  if (HAL_TIM_Base_Init(&htim16) != HAL_OK)
  {
    Error_Handler();
  }

}

// Callback: timer has rolled over
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
{
  if (htim == &htim16 )
  {
      if(gameOn)
      {
          gameOn = 0;
          DisplayEndScreen();
      }
  }

}
```

Glavna zanka

V main funkciji izvedemo inicializacijo naprav in spremenljivk. V funkciji CIrcle_Popper se izvaja igra.

Dokler se igra ne zaključi (gameOn == true) preverjamo, če se je igralec dotaknil zaslona. Če je to prvi dotik nastavimo seed za naključni generator na čas iz časovnika, saj srand(time(null)) ne deluje pravilno. Prikažemo prvi krog. Nastavimo parametre za časovnik. Počistimo zastavico za prekinitev (brez tega se prekinitev sproži takoj) in vklopimo prekinitev.

```
while (*gameOn)
{
  /* Check in polling mode in touch screen the touch status and coordinates */
  /* of touches if touch occurred                                           */
  ts_status = BSP_TS_GetState(0, &TS_State);
  if(TS_State.TouchDetected)
  {
      // Start game after touch; Make first circle and start timer
      if (x_pos == -1 && y_pos == -1){
          // get random seed from timer
          srand(__HAL_TIM_GET_COUNTER(&TIM3Handle));
          Get_Circle(&x_pos,&y_pos, radius);

          // reset timer params
          (htim16).Instance->ARR = 30000; // ~3s
          (htim16).Instance->CNT = (0);

          //clear the interupt bit
          __HAL_TIM_CLEAR_IT(&htim16, TIM_IT_UPDATE);
          // start interupt
          HAL_TIM_Base_Start_IT(&htim16);

          continue;

      }
```

Če to ni prvi dotik preverjamo, če je uporabnik zadel krog. V primeru, da je zadel krog počimo, ponastavimo časovnik in zmanjšamo čas za pok kroga.

```
      /* Get X and Y position of the first touch post calibrated */
      x1 = TS_State.TouchX;
      y1 = TS_State.TouchY;

      if ((y1 > (y_pos - radius)) &&
          (y1 < (y_pos + radius)))
      {
        if ((x1 > (x_pos - radius)) &&
            (x1 < (x_pos + radius)))
        {
            // reset timer
            (htim16).Instance->CNT = (0);
            // speed up the game
            if((htim16).Instance->ARR >= 15000){
                (htim16).Instance->ARR -= 1000;
            } else{
                (htim16).Instance->ARR -= 500;
            }

            Explode_Circle(&x_pos,&y_pos, radius);
            popped++;
      }

    } /* of if(TS_State.TouchDetected) */
```

Ko je igralec prepočasen mu prekinitev ustavi igro in pokaže končni zaslon. Ob kliku na končni zaslon se igra ponovno začne.

```
  HAL_TIM_Base_Stop_IT(&htim16);
  HAL_Delay(300);

  while(!*gameOn){
      ts_status = BSP_TS_GetState(0, &TS_State);
            if(TS_State.TouchDetected)
            {
            *gameOn = 1;
            HAL_Delay(200);
            }
      HAL_Delay(20);
  }
```

Celotna koda

```
/* USER CODE BEGIN Header */
/**
  ******************************************************************************
  * @file           : main.c
  * @brief          : Main program body
  ******************************************************************************
  * @attention
  *
  * Copyright (c) 2022 STMicroelectronics.
  * All rights reserved.
  *
  * This software is licensed under terms that can be found in the LICENSE file
  * in the root directory of this software component.
  * If no LICENSE file comes with this software, it is provided AS-IS.
  *
  ******************************************************************************
  */
/* USER CODE END Header */
/* Includes ------------------------------------------------------------------*/
#include "main.h"
#include "cmsis_os.h"

/* Private includes ----------------------------------------------------------*/
/* USER CODE BEGIN Includes */
//#include "FreeRTOS.h"
#include "stlogo.h"
#include "app_timers.h"

#include "uart.h"
#include "retarget.h"
#include "dma.h"
#include "stdlib.h"


/* USER CODE END Includes */

/* Private variables ---------------------------------------------------------*/
UART_HandleTypeDef                    UART3Handle;
TIM_HandleTypeDef                     TIM3Handle;
/* USER CODE BEGIN PV */
__IO uint32_t ButtonState = 0;
uint32_t x_size;
uint32_t y_size;
uint16_t timer_val_start, timer_val_end;
uint16_t elapsed_1st, elapsed_2nd, elapsed_3rd;
extern TS_Init_t hTS;
extern TS_State_t  TS_State;

TIM_HandleTypeDef htim16;
uint16_t gameOn;
uint16_t popped;
uint32_t ts_status = BSP_ERROR_NONE;

uint8_t time_str1[60];
uint8_t time_str2[40];
```

```c
/* Private function prototypes -----------------------------------------------*/
static void SystemClock_Config(void);



/* USER CODE BEGIN PFP */
static void DisplayStartScreen(void);
/* USER CODE END PFP */


/**
  * @brief TIM16 Initialization Function
  * @param None
  * @retval None
  */
static void MX_TIM16_Init(void)
{
	HAL_NVIC_SetPriority(TIM16_IRQn, TICK_INT_PRIORITY, 0U);
	HAL_NVIC_EnableIRQ(TIM16_IRQn);
	__HAL_RCC_TIM16_CLK_ENABLE();

  htim16.Instance = TIM16;
  htim16.Init.Prescaler = 1000000;
  htim16.Init.CounterMode = TIM_COUNTERMODE_UP;
  // gets set in the game
//  htim16.Init.Period = 30000;
  htim16.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
  htim16.Init.RepetitionCounter = 0;
  htim16.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
  if (HAL_TIM_Base_Init(&htim16) != HAL_OK)
  {
    Error_Handler();
  }

}


static void Clear_LCD()
{
	/* Clear the LCD */
	  UTIL_LCD_SetBackColor(UTIL_LCD_COLOR_BLACK);
	  UTIL_LCD_Clear(UTIL_LCD_COLOR_BLACK);
	  BSP_LCD_FillRect(0, 0, 0, x_size, y_size, UTIL_LCD_COLOR_BLACK);
}

void Get_Circle(int32_t *XPos,int32_t *YPos, int32_t rad)
{
	Clear_LCD();
	*XPos = (rand()%(x_size - 2*rad)) + rad;
	*YPos = (rand()%(y_size - 2*rad)) + rad;

	UTIL_LCD_FillCircle(*XPos,*YPos,rad,UTIL_LCD_COLOR_RED);
}

void Explode_Circle(int32_t *XPos,int32_t *YPos, int32_t rad)
{
	Clear_LCD();
	UTIL_LCD_DrawCircle(*XPos,*YPos,rad + 10,UTIL_LCD_COLOR_RED);
```

```
      HAL_Delay(100);

      Get_Circle(XPos,YPos, rad);

}

static void DisplayEndScreen(void)
{
      Clear_LCD();

// Naslov
  UTIL_LCD_SetFont(&Font24);;
  UTIL_LCD_SetTextColor(UTIL_LCD_COLOR_RED);
  UTIL_LCD_DisplayStringAt(0, y_size/2 - 25 , (uint8_t *)"GAME OVER",
CENTER_MODE);

  // Result
  UTIL_LCD_SetTextColor(UTIL_LCD_COLOR_WHITE);
  UTIL_LCD_SetFont(&Font16);
  sprintf((char* )time_str2, (const char*)"You popped %d circles!", popped);
  UTIL_LCD_DisplayStringAt(0, (y_size/2 + 30), (uint8_t *)time_str2, CENTER_MODE);

  // Rssult
    UTIL_LCD_SetTextColor(UTIL_LCD_COLOR_WHITE);
    UTIL_LCD_SetFont(&Font16);
    sprintf((char* )time_str2, (const char*)"Try again?");
    UTIL_LCD_DisplayStringAt(0, (y_size/2 + 50), (uint8_t *)time_str2,
CENTER_MODE);


}

void Circle_Popper(uint16_t *gameOn)
{

  uint16_t x1, y1;

  uint32_t x_size, y_size;
  int32_t x_pos = -1, y_pos = -1;
  int32_t radius = 20;

  BSP_LCD_GetXSize(0, &x_size);
  BSP_LCD_GetYSize(0, &y_size);
  Clear_LCD();
  ButtonState = 0;
  popped = 0;



  if(ts_status == BSP_ERROR_NONE)
  {

        DisplayStartScreen();

    while (*gameOn)
    {
      /* Check in polling mode in touch screen the touch status and coordinates */
      /* of touches if touch occurred                                           */
```

```
      ts_status = BSP_TS_GetState(0, &TS_State);
      if(TS_State.TouchDetected)
      {
         // Start game after touch; Make first circle and start timer
         if (x_pos == -1 && y_pos == -1){
                // get random seed from timer
                srand(__HAL_TIM_GET_COUNTER(&TIM3Handle));
               Get_Circle(&x_pos,&y_pos, radius);

               // reset timer params
               (htim16).Instance->ARR = 30000; // ~3s
               (htim16).Instance->CNT = (0);

                //clear the interupt bit
                __HAL_TIM_CLEAR_IT(&htim16, TIM_IT_UPDATE);
                // start interupt
                HAL_TIM_Base_Start_IT(&htim16);

                continue;

         }

         /* One or dual touch have been detected          */
         /* Only take into account the first touch so far */

         /* Get X and Y position of the first touch post calibrated */
         x1 = TS_State.TouchX;
         y1 = TS_State.TouchY;

         if ((y1 > (y_pos - radius)) &&
             (y1 < (y_pos + radius)))
         {
           if ((x1 > (x_pos - radius)) &&
               (x1 < (x_pos + radius)))
           {
                // reset timer
                (htim16).Instance->CNT = (0);
                // speed up the game
                if((htim16).Instance->ARR >= 15000){
                       (htim16).Instance->ARR -= 1000;
                } else{
                       (htim16).Instance->ARR -= 500;
                }

                Explode_Circle(&x_pos,&y_pos, radius);
                popped++;
         }

      } /* of if(TS_State.TouchDetected) */


      HAL_Delay(20);
    }
  } /* of if(status == BSP_ERROR_NONE) */
}
  HAL_TIM_Base_Stop_IT(&htim16);
  HAL_Delay(300);

  while(!*gameOn){
```

```
        ts_status = BSP_TS_GetState(0, &TS_State);
              if(TS_State.TouchDetected)
              {
              *gameOn = 1;
              HAL_Delay(200);
              }
        HAL_Delay(20);
  }

}


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

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

  /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
  /* STM32H7xx HAL library initialization:
        - Systick timer is configured by default as source of time base, but user
          can eventually implement his proper time base source (a general purpose
          timer for example or other time source), keeping in mind that Time base
          duration should be kept 1ms since PPP_TIMEOUT_VALUEs are defined and
          handled in milliseconds basis.
        - Set NVIC Group Priority to 4
        - Low Level Initialization
  */
  HAL_Init();



  /* Configure the system clock to 400 MHz */
  SystemClock_Config();

  /* Configure TIM3 timebase */
  Init_TIM3(&TIM3Handle);



  BSP_LED_Init(LED_GREEN);
  BSP_LED_Init(LED_RED);

  BSP_LED_On(LED_GREEN);

  HAL_GPIO_WritePin(GPIOE, GPIO_PIN_3, GPIO_PIN_RESET);

  /* Configure the User push-button in EXTI Mode */
  BSP_PB_Init(BUTTON_USER, BUTTON_MODE_EXTI);

  BSP_LCD_Init(0, LCD_ORIENTATION_LANDSCAPE);
  UTIL_LCD_SetFuncDriver(&LCD_Driver);

//timer innit
  MX_TIM16_Init();
```

```
  // Set LCD size
  BSP_LCD_GetXSize(0, &x_size);
  BSP_LCD_GetYSize(0, &y_size);

       hTS.Width = x_size;
       hTS.Height = y_size;
       hTS.Orientation =TS_SWAP_XY ;
       hTS.Accuracy = 5;


       /* Touchscreen initialization */
       ts_status = BSP_TS_Init(0, &hTS);


//  Display_InitialContent();
  gameOn = 1;
  while(1){
   Circle_Popper(&gameOn);
  }



  /* USER CODE BEGIN WHILE */
  while (1)
  {
  }
  /* USER CODE END 3 */
}

// Callback: timer has rolled over
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
{
  if (htim == &htim16 )
  {
        if(gameOn)
        {
              gameOn = 0;
              DisplayEndScreen();
        }
  }

}



/**
  * @brief  Display main demo messages
  * @param  None
  * @retval None
  */
static void DisplayStartScreen(void)
{

  /* Clear the LCD */
  UTIL_LCD_SetBackColor(UTIL_LCD_COLOR_BLACK);
  UTIL_LCD_Clear(UTIL_LCD_COLOR_BLACK);
  BSP_LCD_FillRect(0, 0, 0, x_size, y_size, UTIL_LCD_COLOR_BLACK);
```

```
// Naslov
  UTIL_LCD_SetFont(&Font24);;
  UTIL_LCD_SetTextColor(UTIL_LCD_COLOR_BLUE);
  UTIL_LCD_DisplayStringAt(0, y_size/2 - 25 , (uint8_t *)"Circle popper",
CENTER_MODE);

  // Navodila
  UTIL_LCD_SetTextColor(UTIL_LCD_COLOR_WHITE);
  UTIL_LCD_SetFont(&Font16);
  sprintf((char* )time_str1, (const char*)"Touch to start. Pop as many as you
can!");
  UTIL_LCD_DisplayStringAt(0, (y_size/2 + 50), (uint8_t *)time_str1, CENTER_MODE);


  //Podpis
  UTIL_LCD_SetFont(&Font16);
  UTIL_LCD_SetTextColor(UTIL_LCD_COLOR_WHITE);
  UTIL_LCD_DisplayStringAt(0, y_size - 20, (uint8_t *)"Rok Svikart", RIGHT_MODE);

}

/**
  * @brief  Check for user input
  * @param  None
  * @retval Input state (1 : active / 0 : Inactive)
  */
uint8_t CheckForUserInput(void)
{
  return ButtonState;
}
/**
* @brief  EXTI line detection callbacks.
* @param  GPIO_Pin: Specifies the pins connected EXTI line
* @retval None
*/
void BSP_PB_Callback(Button_TypeDef Button)
{
 if(Button == BUTTON_USER)
  {
    ButtonState = 1;
  }
}



/**
  * @brief  System Clock Configuration
  *         The system Clock is configured as follow :
  *            System Clock source            = PLL (HSE)
  *            SYSCLK(Hz)                     = 400000000 (Cortex-M7 CPU Clock)
  *            HCLK(Hz)                       = 200000000 (Cortex-M4 CPU, Bus
matrix Clocks)
  *            AHB Prescaler                  = 2
  *            D1 APB3 Prescaler              = 2 (APB3 Clock  100MHz)
  *            D2 APB1 Prescaler              = 2 (APB1 Clock  100MHz)
  *            D2 APB2 Prescaler              = 2 (APB2 Clock  100MHz)
  *            D3 APB4 Prescaler              = 2 (APB4 Clock  100MHz)
  *            HSE Frequency(Hz)              = 25000000
  *            PLL_M                          = 5
```

```
  *            PLL_N                          = 160
  *            PLL_P                          = 2
  *            PLL_Q                          = 4
  *            PLL_R                          = 2
  *            VDD(V)                         = 3.3
  *            Flash Latency(WS)              = 4
  * @param  None
  * @retval None
  */
static void SystemClock_Config(void)
{
  RCC_ClkInitTypeDef RCC_ClkInitStruct;
  RCC_OscInitTypeDef RCC_OscInitStruct;
  HAL_StatusTypeDef ret = HAL_OK;

  /*!< Supply configuration update enable */
  HAL_PWREx_ConfigSupply(PWR_LDO_SUPPLY);

  /* The voltage scaling allows optimizing the power consumption when the device
is
     clocked below the maximum system frequency, to update the voltage scaling
value
     regarding system frequency refer to product datasheet.  */
  __HAL_PWR_VOLTAGESCALING_CONFIG(PWR_REGULATOR_VOLTAGE_SCALE1);

  while(!__HAL_PWR_GET_FLAG(PWR_FLAG_VOSRDY)) {}

  /* Enable HSE Oscillator and activate PLL with HSE as source */
  RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSE;
  RCC_OscInitStruct.HSEState = RCC_HSE_ON;
  RCC_OscInitStruct.HSIState = RCC_HSI_OFF;
  RCC_OscInitStruct.CSIState = RCC_CSI_OFF;
  RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
  RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSE;

  RCC_OscInitStruct.PLL.PLLM = 5;
  RCC_OscInitStruct.PLL.PLLN = 160;
  RCC_OscInitStruct.PLL.PLLFRACN = 0;
  RCC_OscInitStruct.PLL.PLLP = 2;
  RCC_OscInitStruct.PLL.PLLR = 2;
  RCC_OscInitStruct.PLL.PLLQ = 4;

  RCC_OscInitStruct.PLL.PLLVCOSEL = RCC_PLL1VCOWIDE;
  RCC_OscInitStruct.PLL.PLLRGE = RCC_PLL1VCIRANGE_2;
  ret = HAL_RCC_OscConfig(&RCC_OscInitStruct);
  if(ret != HAL_OK)
  {
    Error_Handler();
  }

/* Select PLL as system clock source and configure  bus clocks dividers */
  RCC_ClkInitStruct.ClockType = (RCC_CLOCKTYPE_SYSCLK | RCC_CLOCKTYPE_HCLK |
RCC_CLOCKTYPE_D1PCLK1 | RCC_CLOCKTYPE_PCLK1 | \
                                 RCC_CLOCKTYPE_PCLK2  | RCC_CLOCKTYPE_D3PCLK1);

  RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
  RCC_ClkInitStruct.SYSCLKDivider = RCC_SYSCLK_DIV1;
  RCC_ClkInitStruct.AHBCLKDivider = RCC_HCLK_DIV2;
  RCC_ClkInitStruct.APB3CLKDivider = RCC_APB3_DIV2;
```

```
  RCC_ClkInitStruct.APB1CLKDivider = RCC_APB1_DIV2;
  RCC_ClkInitStruct.APB2CLKDivider = RCC_APB2_DIV2;
  RCC_ClkInitStruct.APB4CLKDivider = RCC_APB4_DIV2;
  ret = HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_4);
  if(ret != HAL_OK)
  {
    Error_Handler();
  }

 /*
  Note : The activation of the I/O Compensation Cell is recommended with
communication  interfaces
          (GPIO, SPI, FMC, QSPI ...)  when  operating at  high frequencies(please
refer to product datasheet)
          The I/O Compensation Cell activation  procedure requires :
        - The activation of the CSI clock
        - The activation of the SYSCFG clock
        - Enabling the I/O Compensation Cell : setting bit[0] of register
SYSCFG_CCCSR
 */

  /*activate CSI clock mondatory for I/O Compensation Cell*/
  __HAL_RCC_CSI_ENABLE() ;

  /* Enable SYSCFG clock mondatory for I/O Compensation Cell */
  __HAL_RCC_SYSCFG_CLK_ENABLE() ;

  /* Enables the I/O Compensation Cell */
  HAL_EnableCompensationCell();
}


/**
  * @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();
  BSP_LED_On(LED_RED);
  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 */
```