Boopit

1.0

Generated by Doxygen 1.10.0

Chapter 1

Data Structure Index

1.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 2

File Index

2.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 3

Data Structure Documentation

3.1 Button Struct Reference

Data Fields

- int x
- int y
- int width
- int height
- char ∗ text
- GLCD_FONT ∗ font
- int input_delay
- int time_pressed

3.1.1 Detailed Description

Definition at line 45 of file display.h.

3.1.2 Field Documentation

3.1.2.1 font

```
GLCD_FONT* font
```

Definition at line 51 of file display.h.

3.1.2.2 height

```
int height
```

Definition at line 49 of file display.h.

3.1.2.3 input_delay

```
int input_delay
```

Definition at line 52 of file display.h.

3.1.2.4 text

```
char* text
```

Definition at line 50 of file display.h.

3.1.2.5 time_pressed

```
int time_pressed
```

Definition at line 53 of file display.h.

3.1.2.6 width

```
int width
```

Definition at line 48 of file display.h.

3.1.2.7 x

```
int x
```

Definition at line 46 of file display.h.

3.1.2.8 y

```
int y
```

Definition at line 47 of file display.h.

The documentation for this struct was generated from the following file:

- Boopit/src/include/display.h

3.2 UserData Struct Reference

Data Fields

- Difficulty difficulty
- int lives
- int baseTime
- int score
- NextScene nextScene

3.2.1 Detailed Description

Definition at line 11 of file userData.h.

3.2.2 Field Documentation

3.2.2.1 baseTime

```
int baseTime
```

Definition at line 14 of file userData.h.

3.2.2.2 difficulty

```
Difficulty difficulty
```

Definition at line 12 of file userData.h.

3.2.2.3 lives

```
int lives
```

Definition at line 13 of file userData.h.

3.2.2.4 nextScene

```
NextScene nextScene
```

Definition at line 16 of file userData.h.

3.2.2.5 score

```
int score
```

Definition at line 15 of file userData.h.

The documentation for this struct was generated from the following file:

- Boopit/src/include/userData.h

Chapter 4

File Documentation

4.1 display.c

```c
#include <string.h>

#include "Board_Touch.h"
#include "GLCD_Config.h"
#include "stm32f7xx_hal.h"

#include "display.h"

extern GLCD_FONT GLCD_Font_6x8;
extern GLCD_FONT GLCD_Font_16x24;

void SystemClock_Config(void) {
    RCC_OscInitTypeDef RCC_OscInitStruct;
    RCC_ClkInitTypeDef RCC_ClkInitStruct;
    /* Enable Power Control clock */
    __HAL_RCC_PWR_CLK_ENABLE();
    /* The voltage scaling allows optimizing the power
    consumption when the device is clocked below the
    maximum system frequency. */
    __HAL_PWR_VOLTAGESCALING_CONFIG(PWR_REGULATOR_VOLTAGE_SCALE1);
    /* Enable HSE Oscillator and activate PLL
    with HSE as source */
    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 = 25;
    RCC_OscInitStruct.PLL.PLLN = 336;
    RCC_OscInitStruct.PLL.PLLP = RCC_PLLP_DIV2;
    RCC_OscInitStruct.PLL.PLLQ = 7;
    HAL_RCC_OscConfig(&RCC_OscInitStruct);
    /* Select PLL as system clock source and configure
    the HCLK, PCLK1 and PCLK2 clocks dividers */
    RCC_ClkInitStruct.ClockType =
        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_DIV2;
    HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_5);
}

void init_display(void) {
    SystemClock_Config(); // Config Clocks
    Touch_Initialize();   // Init Touchscreen
    GLCD_Initialize();    // Init GLCD

    GLCD_ClearScreen();
    GLCD_SetFont(&GLCD_Font_16x24);
    GLCD_SetBackgroundColor(GLCD_COLOR_WHITE);
    GLCD_SetForegroundColor(GLCD_COLOR_BLACK);
};

char debug_buffers[DEBUG_BUFFER_COUNT][DEBUG_BUFFER_SIZE];

void debug_print(void) {
    int buffer;
    if (!DEBUG_MODE) {
        return;
    }
    GLCD_SetFont(&GLCD_Font_6x8);
    for (buffer = 0; buffer < DEBUG_BUFFER_COUNT; buffer++) {
        GLCD_DrawString(6, 8 + (8 + 1) * buffer, debug_buffers[buffer]);
    }
}

void debug_clear(unsigned int buffer) {
    memset(debug_buffers[buffer], 0, DEBUG_BUFFER_SIZE);
}

void debug_clear_all(void) {
    for (int buffer = 0; buffer < DEBUG_BUFFER_COUNT; buffer++) {
        debug_clear(buffer);
    }
}

void draw_button(Button *button) {
    int text_width = button->font->width * strlen(button->text);
    int text_height = button->font->height;
    GLCD_DrawRectangle(button->x, button->y, button->width, button->height);

    GLCD_SetFont(button->font);
    GLCD_DrawString((int)(button->x + button->width / 2) - text_width / 2,
                    (int)(button->y + button->height / 2) - text_height / 2,
                    button->text);
};

bool check_button_press(Button *button, TOUCH_STATE *tsc_state) {
    if (tsc_state->x > button->x && tsc_state->x < button->x + button->width &&
        tsc_state->y > button->y && tsc_state->y < button->y + button->height) {

        int currentTime = HAL_GetTick();

        if (button->time_pressed + button->input_delay < currentTime) {
            button->time_pressed = currentTime;
            return true;
        }
    }
    return false;
};
```

4.2 endScreen.c

```
#include <stdbool.h>
#include <stdio.h>
#include <string.h>

#include "stm32f7xx_hal.h"

#include "display.h"
#include "userData.h"

extern GLCD_FONT GLCD_Font_6x8;
extern GLCD_FONT GLCD_Font_16x24;

static Button replay_button = {(int)(SCREEN_WIDTH * 0.25) - (150 / 2),
                               (int)(SCREEN_HEIGHT * 0.75) - (50 / 2),
                               150,
                               50,
                               "Go again",
                               &GLCD_Font_16x24,
                               0,
                               0};

static Button main_menu_button = {(int)(SCREEN_WIDTH * 0.75) - (150 / 2),
                                  (int)(SCREEN_HEIGHT * 0.75) - (50 / 2),
                                  150,
                                  50,
                                  "Main Menu",
                                  &GLCD_Font_16x24,
                                  150,
                                  0};

static void handle_input(TOUCH_STATE *tsc_state, UserData *userData,
                         bool *inEndScreen) {

    Touch_GetState(tsc_state);
    if (!tsc_state->pressed) {
        return;
    }

    if (check_button_press(&replay_button, tsc_state)) {
        userData->nextScene = GAME;
        handle_difficulty(userData);
        userData->score = 0;
        *inEndScreen = false;
    } else if (check_button_press(&main_menu_button, tsc_state)) {
        userData->nextScene = MAIN_MENU;
        *inEndScreen = false;
    }
}

static void draw_end_screen(int score) {

    char *title = "Game Over!";
    int title_size = strlen(title) * 16;

    GLCD_SetFont(&GLCD_Font_16x24);
    GLCD_DrawString((int)(SCREEN_WIDTH / 2) - (title_size / 2), 50, title);

    char scoreBuffer[256];
    sprintf(scoreBuffer, "Score: %i", score);
    GLCD_DrawString((int)(SCREEN_WIDTH / 2) - (strlen(scoreBuffer) / 2), 120,
                    scoreBuffer);

    draw_button(&replay_button);
    draw_button(&main_menu_button);

    debug_print();
};

void end_screen(UserData *userData) {
    TOUCH_STATE tsc_state;

    bool inEndScreen = true;

    GLCD_ClearScreen();
    debug_clear_all();

    while (inEndScreen) {

        handle_input(&tsc_state, userData, &inEndScreen);

        sprintf(debug_buffers[0], "System Time: %ims   ", HAL_GetTick());
        sprintf(debug_buffers[1], "Touch: %i @ X:%i,Y:%i    ",
                tsc_state.pressed, tsc_state.x, tsc_state.y);
        sprintf(debug_buffers[2], "Difficulty: %i   ", userData->difficulty);
        sprintf(debug_buffers[3], "Score: %i", userData->score);

        draw_end_screen(userData->score);
    }

    GLCD_ClearScreen();
    debug_clear_all();
};
```

4.3 game.c

```
#include <stdbool.h>
#include <stdio.h>
#include <stdlib.h>

#include "Board_LED.h"
#include "Board_Touch.h"
#include "stm32f7xx_hal.h"

#include "display.h"
#include "game.h"
#include "sensor.h"

extern GLCD_FONT GLCD_Font_6x8;
extern GLCD_FONT GLCD_Font_16x24;

extern ADC_HandleTypeDef hadcPhoto;
extern ADC_HandleTypeDef hadcJoyY;

static int last_pressed;
static int previous_task = 0;

static void draw_game_screen(int timeRemaining, int task, int score,
                             int lives) {

    char timeRemainingBuffer[256];
    char taskBuffer[256];
    char scoreBuffer[256];
    char livesBuffer[256];

    GLCD_SetFont(&GLCD_Font_16x24);
    GLCD_DrawString(150, 50, "BOOPIT!");

    sprintf(timeRemainingBuffer, "Remaining time: %.1f  ",
            (float)timeRemaining / 1000);
    GLCD_DrawString(100, 80, timeRemainingBuffer);

    sprintf(taskBuffer, "Task: %s   ", TASK_NAMES[task]);
    GLCD_DrawString(100, 200, taskBuffer);

    sprintf(scoreBuffer, "Score: %i   ", score);
    GLCD_DrawString(75, 150, scoreBuffer);

    sprintf(livesBuffer, "Lives: %i   ", lives);
    GLCD_DrawString(225, 150, livesBuffer);

    debug_print();
}

void play_game(UserData *userData) {
    int startTime = 0, timeCurrent = 0, timeLimit = 0;
    int endTime;
    Task task = previous_task;

    TOUCH_STATE tsc_state;

    bool taskCompleted = false;

    // Game settings
    while (previous_task == task) {
        srand(HAL_GetTick());
        task = (Task)rand() % 5;
    }

    previous_task = task;

    // task = 3;
    timeLimit = userData->baseTime;
    startTime = HAL_GetTick();
    endTime = startTime + timeLimit;

    int input_delay = 0;

    while (timeCurrent < endTime) {

        timeCurrent = HAL_GetTick();

        Touch_GetState(&tsc_state);

        if (last_pressed + input_delay < timeCurrent) {
            last_pressed = timeCurrent;

            switch (task) {
            case TOUCH:
                taskCompleted = touch_sensor_pressed();
                break;
            case PHOTO:
                taskCompleted = photo_sensor_pressed();
                break;
            case BUTTON:
                taskCompleted = button_sensor_pressed();
                break;
            case JOYSTICK:
                taskCompleted = joystick_sensor_pressed();
                break;
            case DISPLAY:
                taskCompleted = tsc_state.pressed;
                break;
            }
        }

        // check completed
        if (taskCompleted) {
            LED_On(0u);
            userData->score++;
            return;

        } else {
            LED_Off(0U);
        }

        sprintf(debug_buffers[0], "System Time: %ims", timeCurrent);
        sprintf(debug_buffers[1], "Touch: %i @ X:%i,Y:%i   ", tsc_state.pressed,
                tsc_state.x, tsc_state.y);
        sprintf(debug_buffers[2], "Lives: %i", userData->lives);
        sprintf(debug_buffers[3], "Task: %i", task);
        sprintf(debug_buffers[4], "Score: %i ", userData->score);

        draw_game_screen(endTime - timeCurrent, task, userData->score,
                         userData->lives);
    }

    userData->lives--;

    if (userData->lives < 1) {
        userData->nextScene = END;
    }
}

```

4.4 Boopit/src/include/display.h File Reference

```
#include <stdbool.h>
#include "Board_GLCD.h"
#include "Board_Touch.h"
```

Data Structures

- struct Button

Macros

- #define SCREEN_WIDTH 480
- #define SCREEN_HEIGHT 272
- #define DEBUG_MODE true

  This macro sets the debug mode.
- #define DEBUG_BUFFER_SIZE 256
- #define DEBUG_BUFFER_COUNT 6

Typedefs

- typedef struct Button **Button**

Functions

- void SystemClock_Config (void)
- void init_display (void)
- void debug_print (void)

  Print out the contents of all the debug buffers onto the screen.
- void debug_clear (unsigned int buffer)
- void debug_clear_all (void)
- void draw_button (Button ∗button)
- bool check_button_press (Button ∗button, TOUCH_STATE ∗tsc_state)

Variables

- char debug_buffers [DEBUG_BUFFER_COUNT][DEBUG_BUFFER_SIZE]

4.4.1 Macro Definition Documentation

4.4.1.1 DEBUG_BUFFER_COUNT

```
#define DEBUG_BUFFER_COUNT 6
```

Definition at line 30 of file display.h.

4.4.1.2 DEBUG_BUFFER_SIZE

```
#define DEBUG_BUFFER_SIZE 256
```

Definition at line 29 of file display.h.

4.4.1.3 DEBUG_MODE

```
#define DEBUG_MODE true
```

This macro sets the debug mode.

Definition at line 27 of file display.h.

4.4.1.4 SCREEN_HEIGHT

```
#define SCREEN_HEIGHT 272
```

Definition at line 13 of file display.h.

4.4.1.5 SCREEN_WIDTH

```
#define SCREEN_WIDTH 480
```

Definition at line 12 of file display.h.

4.4.2 Function Documentation

4.4.2.1 check_button_press()

```
bool check_button_press (
            Button * button,
            TOUCH_STATE * tsc_state )
```

Definition at line 91 of file display.c.

4.4.2.2 debug_clear()

```
void debug_clear (
             unsigned int buffer )
```

Definition at line 70 of file display.c.

4.4.2.3 debug_clear_all()

```
void debug_clear_all (
             void  )
```

Definition at line 74 of file display.c.

4.4.2.4 debug_print()

```
void debug_print (
             void  )
```

Print out the contents of all the debug buffers onto the screen.

Parameters

None	

Return values

None	

Definition at line 59 of file display.c.

4.4.2.5 draw_button()

```
void draw_button (
             Button * button )
```

Definition at line 80 of file display.c.

4.4.2.6 init_display()

```
void init_display (
             void  )
```

Definition at line 46 of file display.c.

4.4.2.7 SystemClock_Config()

```
void SystemClock_Config (
              void  )
```

System Clock Configuration

Definition at line 15 of file display.c.

4.4.3 Variable Documentation

4.4.3.1 debug_buffers

```
char debug_buffers[DEBUG_BUFFER_COUNT][DEBUG_BUFFER_SIZE]  [extern]
```

Definition at line 57 of file display.c.

4.5 display.h

Go to the documentation of this file.

```
#ifndef __DISPLAY
#include <stdbool.h>

#include "Board_GLCD.h"
#include "Board_Touch.h"
#define __DISPLAY

// Constants

#define SCREEN_WIDTH 480
#define SCREEN_HEIGHT 272

// System Config

void SystemClock_Config(void);

void init_display(void);

// Debug

#define DEBUG_MODE true

#define DEBUG_BUFFER_SIZE 256
#define DEBUG_BUFFER_COUNT 6

extern char debug_buffers[DEBUG_BUFFER_COUNT][DEBUG_BUFFER_SIZE];

void debug_print(void);
void debug_clear(unsigned int buffer);
void debug_clear_all(void);

// Widgets

typedef struct Button {
    int x;
    int y;
    int width;
    int height;
    char *text;
    GLCD_FONT *font;
    int input_delay;
    int time_pressed;
} Button;

void draw_button(Button *button);

bool check_button_press(Button *button, TOUCH_STATE *tsc_state);

#endif /* __DISPLAY */
```

4.6 endScreen.h

```
#ifndef __END_SCREEN
#include "userData.h"
#define __END_SCREEN

void end_screen(UserData *userData);

#endif /* __END_SCREEN */
```

4.7 game.h

```
#ifndef __GAME
#include "userData.h"
#define __GAME

typedef enum Task { TOUCH, PHOTO, BUTTON, JOYSTICK, DISPLAY } Task;

static char *TASK_NAMES[] = {"Touch it!", "Hide it!", "Press it!", "Spin it!",
                             "Boop it!"};

void play_game(UserData *userData);

#endif /* __GAME */
```

4.8 mainMenu.h

```
#ifndef __MAIN_MENU
#include "userData.h"
#define __MAIN_MENU

void main_menu(UserData *userData);

#endif /* __MAIN_MENU */
```

4.9 sensor.h

```
#ifndef __SENSOR
#include <stdbool.h>
// #include "stm32f7xx_hal.h"
// #include "stm32f7xx_hal_gpio.h"
#define __SENSOR

// extern ADC_HandleTypeDef hadcPhoto;
// extern ADC_HandleTypeDef hadcJoyY;

bool touch_sensor_pressed(void);
bool photo_sensor_pressed(void);
bool button_sensor_pressed(void);
bool joystick_sensor_pressed(void);

void MX_ADC_Init_Photo(void);
void MX_ADC_Init_JoyY(void);
void MX_GPIO_Init_Photo(void);
void MX_GPIO_Init_JoyY(void);
void SensorInit(void);

#endif /* __SENSOR */
```

4.10 userData.h

```
#ifndef __USERDATA
#define __USERDATA

typedef enum Difficulty { EASY, MEDIUM, HARD, DIFFICULTY_COUNT } Difficulty;

static char *DIFFICULTY_NAMES[] = {"Easy  ", "Medium", "Hard  ",
                                   "Invalid Difficulty"};

typedef enum NextScene { MAIN_MENU, GAME, END } NextScene;
```

```c

typedef struct UserData {
        Difficulty difficulty;
        int lives;
        int baseTime;
        int score;
        NextScene nextScene;
} UserData;

void handle_difficulty(UserData *userData);

#endif /* __USERDATA */
```

4.11 main.c

```c
#include "RTE_Components.h"
#include CMSIS_device_header
#include "rtx_os.h"

#include <stdbool.h>

#include "display.h"
#include "endScreen.h"
#include "game.h"
#include "mainMenu.h"
#include "sensor.h"
#include "userData.h"

// HAL_GetTick replacement
// from https://developer.arm.com/documentation/ka002485/latest/
extern osRtxInfo_t osRtxInfo;
uint32_t HAL_GetTick(void) { return ((uint32_t)osRtxInfo.kernel.tick); }

int main(void) {
    bool running = true;

    Difficulty difficulty = MEDIUM;
    int lives = 0;
    int baseTime = 0;
    int score = 0;
    NextScene nextScene = MAIN_MENU;
    UserData userData = {difficulty, lives, baseTime, score, nextScene};

    init_display();
    SensorInit();

    while (running) {
        switch (userData.nextScene) {
        case MAIN_MENU:
            main_menu(&userData);
            break;
        case GAME:
            play_game(&userData);
            break;
        case END:
            end_screen(&userData);
            break;
        }
    }
}
```

4.12 mainMenu.c

```c
#include <stdbool.h>
#include <stdio.h>
#include <string.h>

#include "stm32f7xx_hal.h"

#include "display.h"
#include "mainMenu.h"
#include "userData.h"

extern GLCD_FONT GLCD_Font_6x8;
extern GLCD_FONT GLCD_Font_16x24;

static Button play_button = {(int)(SCREEN_WIDTH * 0.15) - (100 / 2),
                             (int)(SCREEN_HEIGHT * 0.50) - (50 / 2),
                             100,
                      50,
                      "Play",
                      &GLCD_Font_16x24,
                      0,
                      0};

static Button difficulty_button = {(int)(SCREEN_WIDTH * 0.15) - (100 / 2),
                            (int)(SCREEN_HEIGHT * 0.50) - (50 / 2) + 70,
                            280,
                            50,
                            "Change Difficulty",
                            &GLCD_Font_16x24,
                            150,
                            0};

static void handle_input(TOUCH_STATE *tsc_state, UserData *userData, bool *inMenu) {

    Touch_GetState(tsc_state);
    if (!tsc_state->pressed) {
        return;
    }

    if (check_button_press(&play_button, tsc_state)) {
        handle_difficulty(userData);
        userData->score = 0;
        userData->nextScene = GAME;
        *inMenu = false;

    } else if (check_button_press(&difficulty_button, tsc_state)) {
        if (userData->difficulty < DIFFICULTY_COUNT - 1) {
            userData->difficulty++;
        } else {
            userData->difficulty = 0;
        }
    }
}

static void draw_main_menu(Difficulty *difficulty) {

    char *title = "BOOPIT!";
    int title_size = strlen(title) * 16;

    GLCD_SetFont(&GLCD_Font_16x24);
    GLCD_DrawString((int)(SCREEN_WIDTH / 2) - (title_size / 2), 50, title);

    GLCD_DrawString((int)(SCREEN_WIDTH / 2) - (title_size / 2), 100,
                    DIFFICULTY_NAMES[*difficulty]);

    draw_button(&play_button);
    draw_button(&difficulty_button);

    debug_print();
};

void main_menu(UserData *userData) {
    TOUCH_STATE tsc_state;

    bool inMenu = true;

    GLCD_ClearScreen();
    debug_clear_all();

    while (inMenu) {

        handle_input(&tsc_state, userData, &inMenu);

        sprintf(debug_buffers[0], "System Time: %ims   ", HAL_GetTick());
        sprintf(debug_buffers[1], "Touch: %i @ X:%i,Y:%i    ",
                tsc_state.pressed, tsc_state.x, tsc_state.y);
        sprintf(debug_buffers[2], "Difficulty: %i   ", userData->difficulty);

        draw_main_menu(&userData->difficulty);
    }
    GLCD_ClearScreen();
};
```

4.13 sensor.c

```c
#include <stdbool.h>

#include "Board_LED.h"
#include "stm32f7xx_hal.h"
#include "stm32f7xx_hal_gpio.h"
```

```c


#include "sensor.h"

ADC_HandleTypeDef hadcPhoto;
ADC_HandleTypeDef hadcJoyY;

bool photo_sensor_pressed(void) {
    if (HAL_ADC_GetValue(&hadcPhoto) > 900) {
        return true;
    } else {
        return false;
    }
}
bool button_sensor_pressed(void) {
    if (HAL_GPIO_ReadPin(GPIOC, GPIO_PIN_7)) {
        return false;
    } else {
        return true;
    }
}
bool joystick_sensor_pressed(void) {
    if (HAL_ADC_GetValue(&hadcJoyY) < 100 ||
        HAL_ADC_GetValue(&hadcJoyY) > 1000) {
        return true;
    } else {
        return false;
    }
}

bool touch_sensor_pressed(void) {
    if (HAL_GPIO_ReadPin(GPIOC, GPIO_PIN_6)) {
        return true;
    } else {
        return false;
    }
}

// Photo Resistor
void MX_ADC_Init_Photo(void) {
    ADC_ChannelConfTypeDef sConfig;

    // Enable ADC CLOCK
    //__HAL_RCC_ADC1_CLK_ENABLE();
    //__HAL_RCC_ADC2_CLK_ENABLE();
    __HAL_RCC_ADC3_CLK_ENABLE();

    hadcPhoto.Instance = ADC3; // # Select the ADC (ADC1, ADC2, ADC3)
    hadcPhoto.Init.ClockPrescaler = ADC_CLOCK_SYNC_PCLK_DIV2;
    hadcPhoto.Init.Resolution = ADC_RESOLUTION_12B;
    hadcPhoto.Init.DataAlign = ADC_DATAALIGN_RIGHT;
    hadcPhoto.Init.NbrOfConversion = 3;
    hadcPhoto.Init.ScanConvMode = ENABLE;
    hadcPhoto.Init.ContinuousConvMode = ENABLE;
    hadcPhoto.Init.DiscontinuousConvMode = DISABLE;
    HAL_ADC_Init(&hadcPhoto);
    // configure channal
    sConfig.Rank = 1;
    sConfig.Channel = ADC_CHANNEL_0; // # Select the ADC Channel (ADC_CHANNEL_X)
    sConfig.SamplingTime = ADC_SAMPLETIME_28CYCLES;
    HAL_ADC_ConfigChannel(&hadcPhoto, &sConfig);
    HAL_ADC_Start(&hadcPhoto);
    HAL_ADC_PollForConversion(&hadcPhoto, HAL_MAX_DELAY);
}

GPIO_InitTypeDef GPIO_InitStruct_Photo;
void MX_GPIO_Init_Photo(void) {
    __HAL_RCC_GPIOA_CLK_ENABLE();
    GPIO_InitStruct_Photo.Mode =
        GPIO_MODE_ANALOG;                   // configure to analog input mode
    GPIO_InitStruct_Photo.Pin = GPIO_PIN_0; // #select GPIO GPIO_PIN_X
    GPIO_InitStruct_Photo.Pull = GPIO_NOPULL;
    HAL_GPIO_Init(GPIOA, &GPIO_InitStruct_Photo); // #select GPIO Group
}

// Joystick Y
void MX_ADC_Init_JoyY(void) {
    ADC_ChannelConfTypeDef sConfigJoyY;

    /* Enable ADC CLOCK */
    //__HAL_RCC_ADC1_CLK_ENABLE();
    //__HAL_RCC_ADC2_CLK_ENABLE();
    __HAL_RCC_ADC3_CLK_ENABLE();

    /* Configure the global features of the ADC (Clock, Resolution, Data
    Alignment and number of conversion) */
    hadcJoyY.Instance = ADC3; // # Select the ADC (ADC1, ADC2, ADC3)
    hadcJoyY.Init.ClockPrescaler = ADC_CLOCK_SYNC_PCLK_DIV2;
    hadcJoyY.Init.Resolution = ADC_RESOLUTION_10B;
    hadcJoyY.Init.DataAlign = ADC_DATAALIGN_RIGHT;
    hadcJoyY.Init.NbrOfConversion = 3;
    hadcJoyY.Init.ScanConvMode = ENABLE;
    hadcJoyY.Init.ContinuousConvMode = ENABLE;
    hadcJoyY.Init.DiscontinuousConvMode = DISABLE;
    HAL_ADC_Init(&hadcJoyY);
    // configure channal
    sConfigJoyY.Rank = 3;
    sConfigJoyY.Channel =
        ADC_CHANNEL_7; // # Select the ADC Channel (ADC_CHANNEL_X)
    sConfigJoyY.SamplingTime = ADC_SAMPLETIME_28CYCLES;
    HAL_ADC_ConfigChannel(&hadcJoyY, &sConfigJoyY);
    HAL_ADC_Start(&hadcJoyY);
    HAL_ADC_PollForConversion(&hadcJoyY, HAL_MAX_DELAY);
}

GPIO_InitTypeDef GPIO_InitStructJoyY;
void MX_GPIO_Init_JoyY(void) {
    __HAL_RCC_GPIOF_CLK_ENABLE();
    GPIO_InitStructJoyY.Mode =
        GPIO_MODE_ANALOG;                 // configure to analog input mode
    GPIO_InitStructJoyY.Pin = GPIO_PIN_9; // #select GPIO GPIO_PIN_X
    GPIO_InitStructJoyY.Pull = GPIO_NOPULL;
    HAL_GPIO_Init(GPIOF, &GPIO_InitStructJoyY); // #select GPIO Group
}

void SensorInit(void) {
    // general
    GPIO_InitTypeDef gpio;
    LED_Initialize();
    __HAL_RCC_GPIOC_CLK_ENABLE();

    // analog
    MX_ADC_Init_Photo();
    MX_ADC_Init_JoyY();
    MX_GPIO_Init_Photo();
    MX_GPIO_Init_JoyY();

    HAL_Init();

    // digital
    gpio.Mode = GPIO_MODE_INPUT;
    gpio.Pull = GPIO_PULLDOWN;
    gpio.Speed = GPIO_SPEED_HIGH;

    gpio.Pin = GPIO_PIN_6;
    HAL_GPIO_Init(GPIOC, &gpio);
}
```

4.14 userData.c

```
#include "userData.h"

void handle_difficulty(UserData *userData) {

    switch (userData->difficulty) {
    case EASY:
        userData->lives = 5;
        userData->baseTime = 3000;
        break;
    case MEDIUM:
        userData->lives = 3;
        userData->baseTime = 2000;
        break;
    case HARD:
        userData->lives = 2;
        userData->baseTime = 1500;
        break;
    default:
        break;
    }
}
```

Index