

Boopit

1.0

Generated by Doxygen 1.10.0

1 Data Structure Index	1
1.1 Data Structures	1
2 File Index	3
2.1 File List	3
3 Data Structure Documentation	5
3.1 Button Struct Reference	5
3.1.1 Detailed Description	5
3.1.2 Field Documentation	5
3.1.2.1 font	5
3.1.2.2 height	5
3.1.2.3 input_delay	6
3.1.2.4 text	6
3.1.2.5 time_pressed	6
3.1.2.6 width	6
3.1.2.7 x	6
3.1.2.8 y	6
3.2 UserData Struct Reference	6
3.2.1 Detailed Description	7
3.2.2 Field Documentation	7
3.2.2.1 baseTime	7
3.2.2.2 difficulty	7
3.2.2.3 lives	7
3.2.2.4 nextScene	7
3.2.2.5 score	7
4 File Documentation	9
4.1 display.c	9
4.2 endScreen.c	10
4.3 game.c	11
4.4 Boopit/src/include/display.h File Reference	13
4.4.1 Macro Definition Documentation	14
4.4.1.1 DEBUG_BUFFER_COUNT	14
4.4.1.2 DEBUG_BUFFER_SIZE	14
4.4.1.3 DEBUG_MODE	14
4.4.1.4 SCREEN_HEIGHT	14
4.4.1.5 SCREEN_WIDTH	14
4.4.2 Function Documentation	14
4.4.2.1 check_button_press()	14
4.4.2.2 debug_clear()	15
4.4.2.3 debug_clear_all()	15
4.4.2.4 debug_print()	15

4.4.2.5 draw_button()	15
4.4.2.6 init_display()	15
4.4.2.7 SystemClock_Config()	16
4.4.3 Variable Documentation	16
4.4.3.1 debug_buffers	16
4.5 display.h	16
4.6 endScreen.h	17
4.7 game.h	17
4.8 mainMenu.h	17
4.9 sensor.h	17
4.10 userData.h	17
4.11 main.c	18
4.12 mainMenu.c	18
4.13 sensor.c	19
4.14 userData.c	21
Index	23

Chapter 1

Data Structure Index

1.1 Data Structures

Here are the data structures with brief descriptions:

Button	5
UserData	6

Chapter 2

File Index

2.1 File List

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

Boopit/src/ display.c	9
Boopit/src/ endScreen.c	10
Boopit/src/ game.c	11
Boopit/src/ main.c	18
Boopit/src/ mainMenu.c	18
Boopit/src/ sensor.c	19
Boopit/src/ userData.c	21
Boopit/src/include/ display.h	13
Boopit/src/include/ endScreen.h	17
Boopit/src/include/ game.h	17
Boopit/src/include/ mainMenu.h	17
Boopit/src/include/ sensor.h	17
Boopit/src/include/ userData.h	17

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

```
00001 #include <string.h>
00002
00003 #include "Board_Touch.h"
00004 #include "GLCD_Config.h"
00005 #include "stm32f7xx_hal.h"
00006
00007 #include "display.h"
00008
00009 extern GLCD_FONT GLCD_Font_6x8;
00010 extern GLCD_FONT GLCD_Font_16x24;
00011
00015 void SystemClock_Config(void) {
00016     RCC_OscInitTypeDef RCC_OscInitStruct;
00017     RCC_ClkInitTypeDef RCC_ClkInitStruct;
00018     /* Enable Power Control clock */
00019     __HAL_RCC_PWR_CLK_ENABLE();
00020     /* The voltage scaling allows optimizing the power
00021     consumption when the device is clocked below the
00022     maximum system frequency. */
00023     __HAL_PWR_VOLTAGESCALING_CONFIG(PWR_REGULATOR_VOLTAGE_SCALE1);
00024     /* Enable HSE Oscillator and activate PLL
00025     with HSE as source */
00026     RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSE;
00027     RCC_OscInitStruct.HSEState = RCC_HSE_ON;
00028     RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
00029     RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSE;
00030     RCC_OscInitStruct.PLL.PLLM = 25;
00031     RCC_OscInitStruct.PLL.PLLN = 336;
00032     RCC_OscInitStruct.PLL.PLLP = RCC_PLLP_DIV2;
00033     RCC_OscInitStruct.PLL.PLLQ = 7;
00034     HAL_RCC_OscConfig(&RCC_OscInitStruct);
00035     /* Select PLL as system clock source and configure
00036     the HCLK, PCLK1 and PCLK2 clocks dividers */
00037     RCC_ClkInitStruct.ClockType =
00038         RCC_CLOCKTYPE_SYCLK | RCC_CLOCKTYPE_PCLK1 | RCC_CLOCKTYPE_PCLK2;
00039     RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
00040     RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
00041     RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV4;
00042     RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV2;
00043     HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_5);
00044 }
00045
00046 void init_display(void) {
00047     SystemClock_Config(); // Config Clocks
00048     Touch_Initialize(); // Init Touchscreen
00049     GLCD_Initialize(); // Init GLCD
00050
00051     GLCD_ClearScreen();
00052     GLCD_SetFont(&GLCD_Font_16x24);
00053     GLCD_SetBackgroundColor(GLCD_COLOR_WHITE);
00054     GLCD_SetForegroundColor(GLCD_COLOR_BLACK);
00055 };
00056
00057 char debug_buffers[DEBUG_BUFFER_COUNT][DEBUG_BUFFER_SIZE];
00058
00059 void debug_print(void) {
00060     int buffer;
00061     if (!DEBUG_MODE) {
```

```

00062     return;
00063 }
00064 GLCD_SetFont(&GLCD_Font_6x8);
00065 for (buffer = 0; buffer < DEBUG_BUFFER_COUNT; buffer++) {
00066     GLCD_DrawString(6, 8 + (8 + 1) * buffer, debug_buffers[buffer]);
00067 }
00068 }
00069
00070 void debug_clear(unsigned int buffer) {
00071     memset(debug_buffers[buffer], 0, DEBUG_BUFFER_SIZE);
00072 }
00073
00074 void debug_clear_all(void) {
00075     for (int buffer = 0; buffer < DEBUG_BUFFER_COUNT; buffer++) {
00076         debug_clear(buffer);
00077     }
00078 }
00079
00080 void draw_button(Button *button) {
00081     int text_width = button->font->width * strlen(button->text);
00082     int text_height = button->font->height;
00083     GLCD_DrawRectangle(button->x, button->y, button->width, button->height);
00084
00085     GLCD_SetFont(button->font);
00086     GLCD_DrawString((int)(button->x + button->width / 2) - text_width / 2,
00087                     (int)(button->y + button->height / 2) - text_height / 2,
00088                     button->text);
00089 };
00090
00091 bool check_button_press(Button *button, TOUCH_STATE *tsc_state) {
00092     if (tsc_state->x > button->x && tsc_state->x < button->x + button->width &&
00093         tsc_state->y > button->y && tsc_state->y < button->y + button->height) {
00094
00095         int currentTime = HAL_GetTick();
00096
00097         if (button->time_pressed + button->input_delay < currentTime) {
00098             button->time_pressed = currentTime;
00099             return true;
00100         }
00101     }
00102     return false;
00103 };

```

4.2 endScreen.c

```

00001 #include <stdbool.h>
00002 #include <stdio.h>
00003 #include <string.h>
00004
00005 #include "stm32f7xx_hal.h"
00006
00007 #include "display.h"
00008 #include "userData.h"
00009
00010 extern GLCD_FONT GLCD_Font_6x8;
00011 extern GLCD_FONT GLCD_Font_16x24;
00012
00013 static Button replay_button = {(int)(SCREEN_WIDTH * 0.25) - (150 / 2),
00014                                (int)(SCREEN_HEIGHT * 0.75) - (50 / 2),
00015                                150,
00016                                50,
00017                                "Go again",
00018                                &GLCD_Font_16x24,
00019                                0,
00020                                0};
00021
00022 static Button main_menu_button = {(int)(SCREEN_WIDTH * 0.75) - (150 / 2),
00023                                   (int)(SCREEN_HEIGHT * 0.75) - (50 / 2),
00024                                   150,
00025                                   50,
00026                                   "Main Menu",
00027                                   &GLCD_Font_16x24,
00028                                   150,
00029                                   0};
00030
00031 static void handle_input(TOUCH_STATE *tsc_state, UserData *userData,
00032                          bool *inEndScreen) {
00033
00034     Touch_GetState(tsc_state);
00035     if (!tsc_state->pressed) {
00036         return;
00037     }
00038 }

```

```

00039     if (check_button_press(&replay_button, tsc_state)) {
00040         userData->nextScene = GAME;
00041         handle_difficulty(userData);
00042         userData->score = 0;
00043         *inEndScreen = false;
00044     } else if (check_button_press(&main_menu_button, tsc_state)) {
00045         userData->nextScene = MAIN_MENU;
00046         *inEndScreen = false;
00047     }
00048 }
00049
00050 static void draw_end_screen(int score) {
00051     char *title = "Game Over!";
00052     int title_size = strlen(title) * 16;
00053
00054     GLCD_SetFont(&GLCD_Font_16x24);
00055     GLCD_DrawString((int)(SCREEN_WIDTH / 2) - (title_size / 2), 50, title);
00056
00057     char scoreBuffer[256];
00058     sprintf(scoreBuffer, "Score: %i", score);
00059     GLCD_DrawString((int)(SCREEN_WIDTH / 2) - (strlen(scoreBuffer) / 2), 120,
00060         scoreBuffer);
00061
00062     draw_button(&replay_button);
00063     draw_button(&main_menu_button);
00064
00065     debug_print();
00066 };
00067
00068 void end_screen(UserData *userData) {
00069     TOUCH_STATE tsc_state;
00070
00071     bool inEndScreen = true;
00072
00073     GLCD_ClearScreen();
00074     debug_clear_all();
00075
00076     while (inEndScreen) {
00077         handle_input(&tsc_state, userData, &inEndScreen);
00078
00079         sprintf(debug_buffers[0], "System Time: %ims ", HAL_GetTick());
00080         sprintf(debug_buffers[1], "Touch: %i @ X:%i,Y:%i ",
00081             tsc_state.pressed, tsc_state.X, tsc_state.Y);
00082         sprintf(debug_buffers[2], "Difficulty: %i ", userData->difficulty);
00083         sprintf(debug_buffers[3], "Score: %i", userData->score);
00084
00085         draw_end_screen(userData->score);
00086     }
00087
00088     GLCD_ClearScreen();
00089     debug_clear_all();
00090 };
00091
00092

```

4.3 game.c

```

00001 #include <stdbool.h>
00002 #include <stdio.h>
00003 #include <stdlib.h>
00004
00005 #include "Board_LED.h"
00006 #include "Board_Touch.h"
00007 #include "stm32f7xx_hal.h"
00008
00009 #include "display.h"
00010 #include "game.h"
00011 #include "sensor.h"
00012
00013 extern GLCD_FONT GLCD_Font_6x8;
00014 extern GLCD_FONT GLCD_Font_16x24;
00015
00016 extern ADC_HandleTypeDef hadcPhoto;
00017 extern ADC_HandleTypeDef hadcJoyY;
00018
00019 static int last_pressed;
00020 static int previous_task = 0;
00021
00022 static void draw_game_screen(int timeRemaining, int task, int score,
00023     int lives) {
00024
00025     char timeRemainingBuffer[256];
00026     char taskBuffer[256];

```

```

00027     char scoreBuffer[256];
00028     char livesBuffer[256];
00029
00030     GLCD_SetFont(&GLCD_Font_16x24);
00031     GLCD_DrawString(150, 50, "BOOPIT!");
00032
00033     sprintf(timeRemainingBuffer, "Remaining time: %.1f ",
00034             (float)timeRemaining / 1000);
00035     GLCD_DrawString(100, 80, timeRemainingBuffer);
00036
00037     sprintf(taskBuffer, "Task: %s ", TASK_NAMES[task]);
00038     GLCD_DrawString(100, 200, taskBuffer);
00039
00040     sprintf(scoreBuffer, "Score: %i ", score);
00041     GLCD_DrawString(75, 150, scoreBuffer);
00042
00043     sprintf(livesBuffer, "Lives: %i ", lives);
00044     GLCD_DrawString(225, 150, livesBuffer);
00045
00046     debug_print();
00047 }
00048
00049 void play_game(UserData *userData) {
00050     int startTime = 0, timeCurrent = 0, timeLimit = 0;
00051     int endTime;
00052     Task task = previous_task;
00053
00054     TOUCH_STATE tsc_state;
00055
00056     bool taskCompleted = false;
00057
00058     // Game settings
00059     while (previous_task == task) {
00060         srand(HAL_GetTick());
00061         task = (Task)rand() % 5;
00062     }
00063
00064     previous_task = task;
00065
00066     // task = 3;
00067     timeLimit = userData->baseTime;
00068     startTime = HAL_GetTick();
00069     endTime = startTime + timeLimit;
00070
00071     int input_delay = 0;
00072
00073     while (timeCurrent < endTime) {
00074
00075         timeCurrent = HAL_GetTick();
00076
00077         Touch_GetState(&tsc_state);
00078
00079         if (last_pressed + input_delay < timeCurrent) {
00080             last_pressed = timeCurrent;
00081
00082             switch (task) {
00083             case TOUCH:
00084                 taskCompleted = touch_sensor_pressed();
00085                 break;
00086             case PHOTO:
00087                 taskCompleted = photo_sensor_pressed();
00088                 break;
00089             case BUTTON:
00090                 taskCompleted = button_sensor_pressed();
00091                 break;
00092             case JOYSTICK:
00093                 taskCompleted = joystick_sensor_pressed();
00094                 break;
00095             case DISPLAY:
00096                 taskCompleted = tsc_state.pressed;
00097                 break;
00098             }
00099         }
00100
00101         // check completed
00102         if (taskCompleted) {
00103             LED_On(0u);
00104             userData->score++;
00105             return;
00106         } else {
00107             LED_Off(0U);
00108         }
00109
00110         sprintf(debug_buffers[0], "System Time: %ims", timeCurrent);
00111         sprintf(debug_buffers[1], "Touch: %i @ X:%i,Y:%i ", tsc_state.pressed,
00112                 tsc_state.x, tsc_state.y);

```



```

00114         sprintf(debug_buffers[2], "Lives: %i", userData->lives);
00115         sprintf(debug_buffers[3], "Task: %i", task);
00116         sprintf(debug_buffers[4], "Score: %i ", userData->score);
00117
00118         draw_game_screen(endTime - timeCurrent, task, userData->score,
00119                         userData->lives);
00120     }
00121
00122     userData->lives--;
00123
00124     if (userData->lives < 1) {
00125         userData->nextScene = END;
00126     }
00127 }
00128

```

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.](#)

```
00001 #ifndef __DISPLAY
00002 #include <stdbool.h>
00003
00004 #include "Board_GLCD.h"
00005 #include "Board_Touch.h"
00006 #define __DISPLAY
00007
00010 // Constants
00011
00012 #define SCREEN_WIDTH 480
00013 #define SCREEN_HEIGHT 272
00014
00015 // System Config
00016
00017 void SystemClock_Config(void);
00018
00019 void init_display(void);
00020
00021 // Debug
00022
00027 #define DEBUG_MODE true
00028
00029 #define DEBUG_BUFFER_SIZE 256
00030 #define DEBUG_BUFFER_COUNT 6
00031
00032 extern char debug_buffers[DEBUG_BUFFER_COUNT][DEBUG_BUFFER_SIZE];
00033
00039 void debug_print(void);
00040 void debug_clear(unsigned int buffer);
00041 void debug_clear_all(void);
00042
00043 // Widgets
00044
00045 typedef struct Button {
00046     int x;
00047     int y;
00048     int width;
00049     int height;
00050     char *text;
00051     GLCD_FONT *font;
00052     int input_delay;
00053     int time_pressed;
00054 } Button;
00055
00056 void draw_button(Button *button);
00057
00058 bool check_button_press(Button *button, TOUCH_STATE *tsc_state);
00059
00060 #endif /* __DISPLAY */
```

4.6 endScreen.h

```
00001 #ifndef __END_SCREEN
00002 #include "userData.h"
00003 #define __END_SCREEN
00004
00005 void end_screen(UserData *userData);
00006
00007 #endif /* __END_SCREEN */
```

4.7 game.h

```
00001 #ifndef __GAME
00002 #include "userData.h"
00003 #define __GAME
00004
00005 typedef enum Task { TOUCH, PHOTO, BUTTON, JOYSTICK, DISPLAY } Task;
00006
00007 static char *TASK_NAMES[] = {"Touch it!", "Hide it!", "Press it!", "Spin it!",
00008                               "Boop it!"};
00009
00010 void play_game(UserData *userData);
00011
00012 #endif /* __GAME */
```

4.8 mainMenu.h

```
00001 #ifndef __MAIN_MENU
00002 #include "userData.h"
00003 #define __MAIN_MENU
00004
00005 void main_menu(UserData *userData);
00006
00007 #endif /* __MAIN_MENU */
```

4.9 sensor.h

```
00001 #ifndef __SENSOR
00002 #include <stdbool.h>
00003 // #include "stm32f7xx_hal.h"
00004 // #include "stm32f7xx_hal_gpio.h"
00005 #define __SENSOR
00006
00007 // extern ADC_HandleTypeDef hadcPhoto;
00008 // extern ADC_HandleTypeDef hadcJoyY;
00009
00010 bool touch_sensor_pressed(void);
00011 bool photo_sensor_pressed(void);
00012 bool button_sensor_pressed(void);
00013 bool joystick_sensor_pressed(void);
00014
00015 void MX_ADC_Init_Photo(void);
00016 void MX_ADC_Init_JoyY(void);
00017 void MX_GPIO_Init_Photo(void);
00018 void MX_GPIO_Init_JoyY(void);
00019 void SensorInit(void);
00020
00021 #endif /* __SENSOR */
```

4.10 userData.h

```
00001 #ifndef __USERDATA
00002 #define __USERDATA
00003
00004 typedef enum Difficulty { EASY, MEDIUM, HARD, DIFFICULTY_COUNT } Difficulty;
00005
00006 static char *DIFFICULTY_NAMES[] = {"Easy ", "Medium", "Hard ",
00007                                     "Invalid Difficulty"};
00008
00009 typedef enum NextScene { MAIN_MENU, GAME, END } NextScene;
```

```

00010
00011 typedef struct UserData {
00012     Difficulty difficulty;
00013     int lives;
00014     int baseTime;
00015     int score;
00016     NextScene nextScene;
00017 } UserData;
00018
00019 void handle_difficulty(UserData *userData);
00020
00021 #endif /* __USERDATA */

```

4.11 main.c

```

00001 #include "RTE_Components.h"
00002 #include CMSIS_device_header
00003 #include "rtx_os.h"
00004
00005 #include <stdbool.h>
00006
00007 #include "display.h"
00008 #include "endScreen.h"
00009 #include "game.h"
00010 #include "mainMenu.h"
00011 #include "sensor.h"
00012 #include "userData.h"
00013
00014 // HAL_GetTick replacement
00015 // from https://developer.arm.com/documentation/ka002485/latest/
00016 extern osRtxInfo_t osRtxInfo;
00017 uint32_t HAL_GetTick(void) { return ((uint32_t)osRtxInfo.kernel.tick); }
00018
00019 int main(void) {
00020     bool running = true;
00021
00022     Difficulty difficulty = MEDIUM;
00023     int lives = 0;
00024     int baseTime = 0;
00025     int score = 0;
00026     NextScene nextScene = MAIN_MENU;
00027     UserData userData = {difficulty, lives, baseTime, score, nextScene};
00028
00029     init_display();
00030     SensorInit();
00031
00032     while (running) {
00033         switch (userData.nextScene) {
00034             case MAIN_MENU:
00035                 main_menu(&userData);
00036                 break;
00037             case GAME:
00038                 play_game(&userData);
00039                 break;
00040             case END:
00041                 end_screen(&userData);
00042                 break;
00043         }
00044     }
00045 }

```

4.12 mainMenu.c

```

00001 #include <stdbool.h>
00002 #include <stdio.h>
00003 #include <string.h>
00004
00005 #include "stm32f7xx_hal.h"
00006
00007 #include "display.h"
00008 #include "mainMenu.h"
00009 #include "userData.h"
00010
00011 extern GLCD_FONT GLCD_Font_6x8;
00012 extern GLCD_FONT GLCD_Font_16x24;
00013
00014 static Button play_button = {(int)(SCREEN_WIDTH * 0.15) - (100 / 2),
00015                             (int)(SCREEN_HEIGHT * 0.50) - (50 / 2),
00016                             100,

```

```

00017         50,
00018         "Play",
00019         &GLCD_Font_16x24,
00020         0,
00021         0);
00022
00023 static Button difficulty_button = {(int)(SCREEN_WIDTH * 0.15) - (100 / 2),
00024                                   (int)(SCREEN_HEIGHT * 0.50) - (50 / 2) + 70,
00025                                   280,
00026                                   50,
00027                                   "Change Difficulty",
00028                                   &GLCD_Font_16x24,
00029                                   150,
00030                                   0};
00031
00032 static void handle_input(TOUCH_STATE *tsc_state, UserData *userData, bool *inMenu) {
00033
00034     Touch_GetState(tsc_state);
00035     if (!tsc_state->pressed) {
00036         return;
00037     }
00038
00039     if (check_button_press(&play_button, tsc_state)) {
00040         handle_difficulty(userData);
00041         userData->score = 0;
00042         userData->nextScene = GAME;
00043         *inMenu = false;
00044
00045     } else if (check_button_press(&difficulty_button, tsc_state)) {
00046         if (userData->difficulty < DIFFICULTY_COUNT - 1) {
00047             userData->difficulty++;
00048         } else {
00049             userData->difficulty = 0;
00050         }
00051     }
00052 }
00053
00054 static void draw_main_menu(Difficulty *difficulty) {
00055
00056     char *title = "BOOPIT!";
00057     int title_size = strlen(title) * 16;
00058
00059     GLCD_SetFont(&GLCD_Font_16x24);
00060     GLCD_DrawString((int)(SCREEN_WIDTH / 2) - (title_size / 2), 50, title);
00061
00062     GLCD_DrawString((int)(SCREEN_WIDTH / 2) - (title_size / 2), 100,
00063                     DIFFICULTY_NAMES[*difficulty]);
00064
00065     draw_button(&play_button);
00066     draw_button(&difficulty_button);
00067
00068     debug_print();
00069 };
00070
00071 void main_menu(UserData *userData) {
00072     TOUCH_STATE tsc_state;
00073
00074     bool inMenu = true;
00075
00076     GLCD_ClearScreen();
00077     debug_clear_all();
00078
00079     while (inMenu) {
00080
00081         handle_input(&tsc_state, userData, &inMenu);
00082
00083         sprintf(debug_buffers[0], "System Time: %ims ", HAL_GetTick());
00084         sprintf(debug_buffers[1], "Touch: %i @ X:%i,Y:%i ",
00085                 tsc_state.pressed, tsc_state.x, tsc_state.y);
00086         sprintf(debug_buffers[2], "Difficulty: %i ", userData->difficulty);
00087
00088         draw_main_menu(&userData->difficulty);
00089     }
00090     GLCD_ClearScreen();
00091 };

```

4.13 sensor.c

```

00001 #include <stdbool.h>
00002
00003 #include "Board_LED.h"
00004 #include "stm32f7xx_hal.h"
00005 #include "stm32f7xx_hal_gpio.h"

```

```

00006
00007
00008 #include "sensor.h"
00009
00010 ADC_HandleTypeDef hadcPhoto;
00011 ADC_HandleTypeDef hadcJoyY;
00012
00013 bool photo_sensor_pressed(void) {
00014     if (HAL_ADC_GetValue(&hadcPhoto) > 900) {
00015         return true;
00016     } else {
00017         return false;
00018     }
00019 }
00020 bool button_sensor_pressed(void) {
00021     if (HAL_GPIO_ReadPin(GPIOC, GPIO_PIN_7)) {
00022         return false;
00023     } else {
00024         return true;
00025     }
00026 }
00027 bool joystick_sensor_pressed(void) {
00028     if (HAL_ADC_GetValue(&hadcJoyY) < 100 ||
00029         HAL_ADC_GetValue(&hadcJoyY) > 1000) {
00030         return true;
00031     } else {
00032         return false;
00033     }
00034 }
00035
00036 bool touch_sensor_pressed(void) {
00037     if (HAL_GPIO_ReadPin(GPIOC, GPIO_PIN_6)) {
00038         return true;
00039     } else {
00040         return false;
00041     }
00042 }
00043
00044 // Photo Resistor
00045 void MX_ADC_Init_Photo(void) {
00046     ADC_ChannelConfTypeDef sConfig;
00047
00048     // Enable ADC CLOCK
00049     __HAL_RCC_ADC1_CLK_ENABLE();
00050     __HAL_RCC_ADC2_CLK_ENABLE();
00051     __HAL_RCC_ADC3_CLK_ENABLE();
00052
00053     hadcPhoto.Instance = ADC3; // # Select the ADC (ADC1, ADC2, ADC3)
00054     hadcPhoto.Init.ClockPrescaler = ADC_CLOCK_SYNC_PCLK_DIV2;
00055     hadcPhoto.Init.Resolution = ADC_RESOLUTION_12B;
00056     hadcPhoto.Init.DataAlign = ADC_DATAALIGN_RIGHT;
00057     hadcPhoto.Init.NbrOfConversion = 3;
00058     hadcPhoto.Init.ScanConvMode = ENABLE;
00059     hadcPhoto.Init.ContinuousConvMode = ENABLE;
00060     hadcPhoto.Init.DiscontinuousConvMode = DISABLE;
00061     HAL_ADC_Init(&hadcPhoto);
00062     // configure channel
00063     sConfig.Rank = 1;
00064     sConfig.Channel = ADC_CHANNEL_0; // # Select the ADC Channel (ADC_CHANNEL_X)
00065     sConfig.SamplingTime = ADC_SAMPLETIME_28CYCLES;
00066     HAL_ADC_ConfigChannel(&hadcPhoto, &sConfig);
00067     HAL_ADC_Start(&hadcPhoto);
00068     HAL_ADC_PollForConversion(&hadcPhoto, HAL_MAX_DELAY);
00069 }
00070
00071
00072
00073 GPIO_InitTypeDef GPIO_InitStruct_Photo;
00074 void MX_GPIO_Init_Photo(void) {
00075     __HAL_RCC_GPIOA_CLK_ENABLE();
00076     GPIO_InitStruct_Photo.Mode =
00077         GPIO_MODE_ANALOG; // configure to analog input mode
00078     GPIO_InitStruct_Photo.Pin = GPIO_PIN_0; // #select GPIO GPIO_PIN_X
00079     GPIO_InitStruct_Photo.Pull = GPIO_NOPULL;
00080     HAL_GPIO_Init(GPIOA, &GPIO_InitStruct_Photo); // #select GPIO Group
00081 }
00082
00083 // Joystick Y
00084 void MX_ADC_Init_JoyY(void) {
00085     ADC_ChannelConfTypeDef sConfigJoyY;
00086
00087     /* Enable ADC CLOCK */
00088     __HAL_RCC_ADC1_CLK_ENABLE();
00089     __HAL_RCC_ADC2_CLK_ENABLE();
00090     __HAL_RCC_ADC3_CLK_ENABLE();
00091
00092     /* Configure the global features of the ADC (Clock, Resolution, Data
00093     Alignment and number of conversion) */
00094     hadcJoyY.Instance = ADC3; // # Select the ADC (ADC1, ADC2, ADC3)

```



```

00095     hadcJoyY.Init.ClockPrescaler = ADC_CLOCK_SYNC_PCLK_DIV2;
00096     hadcJoyY.Init.Resolution = ADC_RESOLUTION_10B;
00097     hadcJoyY.Init.DataAlign = ADC_DATAALIGN_RIGHT;
00098     hadcJoyY.Init.NbrOfConversion = 3;
00099     hadcJoyY.Init.ScanConvMode = ENABLE;
00100     hadcJoyY.Init.ContinuousConvMode = ENABLE;
00101     hadcJoyY.Init.DiscontinuousConvMode = DISABLE;
00102     HAL_ADC_Init(&hadcJoyY);
00103     // configure channel
00104     sConfigJoyY.Rank = 3;
00105     sConfigJoyY.Channel =
00106         ADC_CHANNEL_7; // # Select the ADC Channel (ADC_CHANNEL_X)
00107     sConfigJoyY.SamplingTime = ADC_SAMPLETIME_28CYCLES;
00108     HAL_ADC_ConfigChannel(&hadcJoyY, &sConfigJoyY);
00109     HAL_ADC_Start(&hadcJoyY);
00110     HAL_ADC_PollForConversion(&hadcJoyY, HAL_MAX_DELAY);
00111 }
00112
00113 GPIO_InitTypeDef GPIO_InitStructJoyY;
00114 void MX_GPIO_Init_JoyY(void) {
00115     __HAL_RCC_GPIOF_CLK_ENABLE();
00116     GPIO_InitStructJoyY.Mode =
00117         GPIO_MODE_ANALOG; // configure to analog input mode
00118     GPIO_InitStructJoyY.Pin = GPIO_PIN_9; // #select GPIO GPIO_PIN_X
00119     GPIO_InitStructJoyY.Pull = GPIO_NOPULL;
00120     HAL_GPIO_Init(GPIOF, &GPIO_InitStructJoyY); // #select GPIO Group
00121 }
00122
00123 void SensorInit(void) {
00124     // general
00125     GPIO_InitTypeDef gpio;
00126     LED_Initialize();
00127     __HAL_RCC_GPIOC_CLK_ENABLE();
00128
00129     // analog
00130     MX_ADC_Init_Photo();
00131     MX_ADC_Init_JoyY();
00132     MX_GPIO_Init_Photo();
00133     MX_GPIO_Init_JoyY();
00134
00135     HAL_Init();
00136
00137     // digital
00138     gpio.Mode = GPIO_MODE_INPUT;
00139     gpio.Pull = GPIO_PULLDOWN;
00140     gpio.Speed = GPIO_SPEED_HIGH;
00141
00142     gpio.Pin = GPIO_PIN_6;
00143     HAL_GPIO_Init(GPIOC, &gpio);
00144 }

```

4.14 userData.c

```

00001 #include "userData.h"
00002
00003 void handle_difficulty(UserData *userData) {
00004
00005     switch (userData->difficulty) {
00006     case EASY:
00007         userData->lives = 5;
00008         userData->baseTime = 3000;
00009         break;
00010     case MEDIUM:
00011         userData->lives = 3;
00012         userData->baseTime = 2000;
00013         break;
00014     case HARD:
00015         userData->lives = 2;
00016         userData->baseTime = 1500;
00017         break;
00018     default:
00019         break;
00020     }
00021 }

```


Index

- baseTime
 - UserData, 7
- Boopit/src/display.c, 9
- Boopit/src/endScreen.c, 10
- Boopit/src/game.c, 11
- Boopit/src/include/display.h, 13, 16
- Boopit/src/include/endScreen.h, 17
- Boopit/src/include/game.h, 17
- Boopit/src/include/mainMenu.h, 17
- Boopit/src/include/sensor.h, 17
- Boopit/src/include/userData.h, 17
- Boopit/src/main.c, 18
- Boopit/src/mainMenu.c, 18
- Boopit/src/sensor.c, 19
- Boopit/src/userData.c, 21
- Button, 5
 - font, 5
 - height, 5
 - input_delay, 5
 - text, 6
 - time_pressed, 6
 - width, 6
 - x, 6
 - y, 6
- check_button_press
 - display.h, 14
- DEBUG_BUFFER_COUNT
 - display.h, 14
- DEBUG_BUFFER_SIZE
 - display.h, 14
- debug_buffers
 - display.h, 16
- debug_clear
 - display.h, 14
- debug_clear_all
 - display.h, 15
- DEBUG_MODE
 - display.h, 14
- debug_print
 - display.h, 15
- difficulty
 - UserData, 7
- display.h
 - check_button_press, 14
 - DEBUG_BUFFER_COUNT, 14
 - DEBUG_BUFFER_SIZE, 14
 - debug_buffers, 16
 - debug_clear, 14
 - debug_clear_all, 15
 - DEBUG_MODE, 14
 - debug_print, 15
 - SCREEN_HEIGHT, 14
 - SCREEN_WIDTH, 14
 - SystemClock_Config, 15
 - draw_button, 15
 - font, 5
 - height, 5
 - init_display, 15
 - input_delay, 5
 - lives, 7
 - nextScene, 7
 - score, 7
 - text, 6
 - time_pressed, 6
 - width, 6
- debug_clear_all, 15
- DEBUG_MODE, 14
- debug_print, 15
- SCREEN_HEIGHT, 14
- SCREEN_WIDTH, 14
- SystemClock_Config, 15
- draw_button
 - display.h, 15
- font
 - Button, 5
- height
 - Button, 5
- init_display
 - display.h, 15
- input_delay
 - Button, 5
- lives
 - UserData, 7
- nextScene
 - UserData, 7
- score
 - UserData, 7
- SCREEN_HEIGHT
 - display.h, 14
- SCREEN_WIDTH
 - display.h, 14
- SystemClock_Config
 - display.h, 15
- text
 - Button, 6
- time_pressed
 - Button, 6
- UserData, 6
 - baseTime, 7
 - difficulty, 7
 - lives, 7
 - nextScene, 7
 - score, 7
- width
 - Button, 6

x

Button, [6](#)

y

Button, [6](#)