Ryerson University

Faculty of Engineering & Architectural Science

Course Number	COE718
Course Title	Embedded Systems
Semester/Year	Fall 2024
Instructor	Dr. G. Khan
Section No.	01

Lab/Tut Assignment NO.	Project

Assignment Title	**Media Center: Final Report**

Name	Student ID	Signature*
Dexter Ryan Floreza	500946679	DRF

(Note: remove the first 4 digits from your student ID)

By signing above you attest that you have contributed to this submission and confirm that all work you have contributed to this submission is your own work. Any suspicion of copying or plagiarism in this work will result in an investigation of Academic Misconduct and may result in a "0" on the work, an "F" in the course, or possibly more severe penalties, as well as a Disciplinary Notice on your academic record under the Student Code of Academic Conduct, which can be found online at: www.ryerson.ca/senate/current/pol60.pdf

COE718 FINAL PROJECT

Dexter Ryan Floreza
Toronto Metropolitan University
Faculty of Engineering and Architectural Science
Toronto, Canada
dexter.floreza@torontomu.ca

***Abstract*—**
***Index Terms*—embedded systems, Keil, MCB1700, multimedia systems**

I. Abstract

This project focuses on designing a media center using the Keil MCB1700 evaluation board, which features the NXP LPC1768 microcontroller, and programming it with the Keil uVision5 IDE. The media center will provide the following functionalities: image viewer, music player, and a gaming hub. With regards to hardware, the joystick and buttons on the board will serve as the interface for navigating the graphical user interface (GUI).

II. Introduction

A. A Gentle Introduction to Embedded Systems

In today's world, small electronic devices are integral to nearly every aspect of human life. From kitchen appliances to entertainment centres, to military technologies used for protecting our nation, we are surrounded by a digital environment powered by logic processors. These processors automate tasks, solve problems, and enhance convenience for users.

At the core of many of these devices are embedded systems, which combine hardware and software to perform specific, programmed tasks. Unlike general-purpose computers, embedded systems are designed for efficiency, with limited memory and processing power. They consist of a microprocessor, memory, and communication buses to process and store data efficiently. Embedded systems may operate independently or interact with other devices and networks, making them versatile solutions for countless applications.

Embedded systems are classified based on their responsiveness to time-critical tasks, including hard real-time systems and soft real-time systems. Systems like missile guidance or automotive anti-lock brakes demand immediate responses and exemplify the hard real-time category, where delays can have serious consequences.

The ARM Cortex-M3 processor, widely used in embedded systems, features a Harvard architecture that separates memory interfaces, improving efficiency and performance. This processor strikes a balance between high performance and low cost, making it ideal for applications in smartphones, vehicles, and IoT devices. For this project, the ARM Cortex-M3 processor will be employed to develop a Media Center.

The Media Center will offer three primary features: a photo gallery for viewing images, an audio system to play sound via USB with volume control, and interactive games navigable through a joystick. These functionalities will be developed in the C programming language using the Keil uVision IDE.

III. Past Work

This project will use the **NXP LPC1768 microcontroller**, which is part of the **Keil MCB1700 evaluation board**. Key hardware features include the **ADC channel AIN2** for volume control and a **joystick** for navigating the graphical user interface (GUI), enabling user interaction with the media center.

Development will be carried out using the **Keil uVision5 IDE**. A variety of source files will support the project, including `ADC.c`, `ADC.h`, `Blinky.c`, `GLCD.h`, `IRQ.c`, `KBD.c`, `KBD.h`, `LED.c`, and `LED.h`. Additionally, **GIMP software** has been used to convert **.jpg images into C files** for integration into the system.

Previously developed programs like `ADC.c`, `Blinky.c`, `KBD.c`, and `LED.c` have been used for tasks such as adjusting voltage via a potentiometer and navigating menus with the joystick. These existing functionalities will be adapted and incorporated into the features of the media center for this project.

IV. Methodology

The media center project incorporates three main features: an **Image Gallery**, a **Media Player**, and the classic **Snake Game**. Each feature is designed to utilize the NXP LPC1768 microcontroller on the Keil MCB1700 board, with functionality navigated through the joystick and volume adjustments made via the potentiometer.

The **Image Gallery** showcases a collection of three Pokémon images: **Arceus**, **Pikachu**, and **Plusle & Minun**. Images were prepared using GIMP, resized to fit the LCD display at 150x112 pixels, rotated 180° for proper orientation, and exported as C source files. The gallery uses an array to iterate through and display images, allowing users to scroll through them by toggling the joystick in the up and downwards directions. The user can exit from the gallery by pressing the buttons to the side of the screen.

The **Media Player** plays the **Verdanturf Town theme**, with audio streamed directly from the NXP LPC1768 chip.

The potentiometer script was repurposed to implement volume control, enabling users to adjust the audio level seamlessly. Minor modifications were made to the USB audio script, such as renaming the main function to `usbmain()` for compatibility with the overall program.

The **Snake Game** offers an interactive experience where players control a snake using the joystick, navigating the screen to collect food items while avoiding collisions with the snake's own tail or the screen's edges. Game logic methods were implemented to handle collision detection and frame transitions, ensuring smooth gameplay and an engaging user experience. ASCII characters were used in the design of the snake.

To tie everything together, a main menu interface was developed, with a `selector()` method to navigate between features. The joystick is essential for selecting options, and users can exit most features by pressing down on the joystick or clicking it. However, for the Snake Game, the exit functionality had some difficulty during gameplay, as the downward joystick movement is part of the controls. This integration of hardware and software delivers a cohesive, multifunctional media center.

V. Design

This section will explain and justify the methodology that was used for ipmlementing each feature of the media centre. The `selector()` method allows users to navigate the main menu upon startup.

A. Photo Gallery

The **Photo Gallery** implementation relied on the `photo_gallery()` function to manage the display and navigation of images. This function begins by setting up the display through the `LCD_Template()` method when transitioning to the Photo Gallery from another section. The function uses joystick inputs, captured via the `get_button()` function, to update the `selector` variable through the `side_ways_selector()` function. The `selector` determines which image is displayed, and the `GLCD_Bitmap()` function is employed to render the corresponding image from the `images_for_photo_gallery` array onto the screen. Navigation through the images is achieved using the joystick's directional inputs, while a button press allows the user to exit the Photo Gallery and return to the home screen. This approach ensures that the gallery is intuitive, responsive, and visually effective without requiring a separate `showGallery()` function.

B. MP3 Player

The **Media Player** functionality is implemented through the `mp3_music_player()` function, which manages the display and audio playback. Upon activation, the function utilizes the `LCD_Template()` to prepare the screen. It then displays relevant text, such as "MUSIC STREAMING!" and "NOW PLAYING: VERDANTURF TOWN," using the `GLCD_DisplayString()` function. Additionally, graphical elements, including a Pokeball and Ash & Pikachu image, are rendered on the screen using the `DisplayPokeball()` and `DisplayAsh()` functions. Audio playback is handled by the `stream_music()` function, which streams the Verdanturf Town theme directly from the LPC1768 microcontroller. Once playback is complete or the user exits, the function transitions back to the home screen. This design ensures an engaging and visually appealing music player experience.

C. Game: Snake

The **Snake Game** is implemented through a series of functions that handle setup, gameplay, and post-game functionality, providing a complete and engaging user experience. The game begins with the `game_center()` function, which immediately transitions to the `snake_main()` function to initiate gameplay. A pre-game screen is displayed via the `pre_game_screen()` function, outlining basic controls and game rules. Once the user clicks the joystick to start, the game begins, rendering the playing area and initializing the snake's starting position.

The gameplay logic, defined in the `snake_game()` function, involves real-time movement and collision detection. The snake's movement is controlled by joystick inputs, and its position is updated dynamically using the `update_snake()` function. The game uses collision detection to determine when the snake collides with itself or the screen borders, ending the game if such an event occurs. Additionally, the `generate_food()` function ensures that food appears in valid positions on the board, while a toggle mechanism alternates the food's display between two symbols ("@" and "#"), adding visual variety.

The snake grows in length as it consumes food, and the `update_snake()` function dynamically updates its appearance on the screen, with distinct visual elements for the head, body links, and tail. The game's score increases with each piece of food eaten, and upon game termination, the `post_game_screen()` function displays the player's score and provides options to restart or return to the main menu. This comprehensive implementation combines real-time user input, dynamic rendering, and game logic to deliver a classic Snake Game experience, tailored for the MCB1700 evaluation board.

VI. Results

A. Main Menu

Upon pressing the red button on the MCB1700 board, the user will be able to start up the machine and see the main menu. The following image, as seen in Figure 1, should be visible.

The user is able to navigate the main menu after startup by using the joystick. The user can click the joystick to explore the different menu items listed above. Upon going to each item, that item will be highlighted yellow.

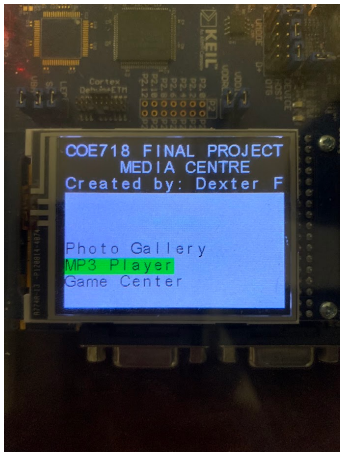

Fig. 1. The Main Menu that the user sees upon startup.

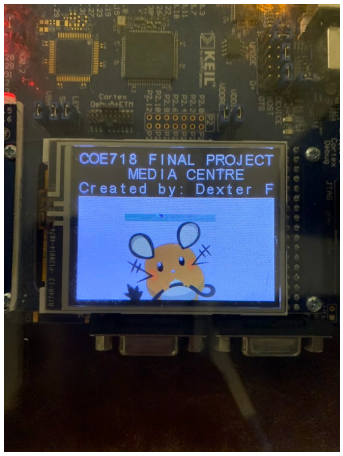

Fig. 2. The first photo in the photo gallery, an image of the Pokémon, Dedenne.

B. Photo Gallery

From a results perspective, the **Photo Gallery** successfully provides an interactive and visually appealing feature in the media center. The implementation allows users to navigate through multiple images, such as Pokémon characters, using the joystick. The transition between images is smooth and responsive, showcasing the effectiveness of the `side_ways_selector()` function in updating the displayed image based on user input.

Each image is rendered clearly on the LCD screen, demonstrating the proper preprocessing and conversion of image files into C-compatible bitmap arrays. This confirms the robustness of the external image preparation process (e.g., resizing, flipping, and formatting with GIMP). The accurate display of images without distortion highlights the compatibility of the processed images with the MCB1700 evaluation board.

User interaction with the Photo Gallery is straightforward and intuitive. Moving the joystick left or right cycles through the gallery, and pressing a button exits back to the main menu. This functionality aligns with the design goals of creating a user-friendly interface. The fluidity of navigation and the ease of transitioning between images reflect the reliability of the joystick-based input handling and the efficiency of the `photo_gallery()` function. Although, there is an issue with the photo display, as it shows a bit of the previous image upon scrolling to the next image.

In summary, the Photo Gallery meets all intended objectives, providing a functional, interactive feature that integrates seamlessly with the media center. The results demonstrate the successful application of hardware-software integration, precise image rendering, and responsive user input handling, making the Photo Gallery a standout element of the media center.

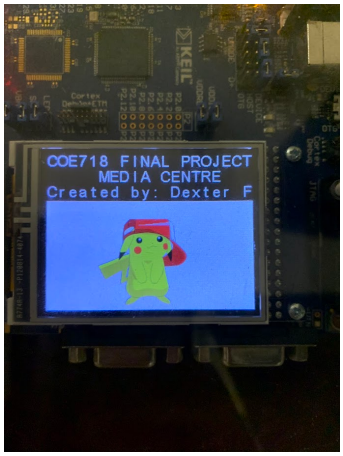

Fig. 3. The second photo in the gallery, an image of the Pokémon, Pikachu.

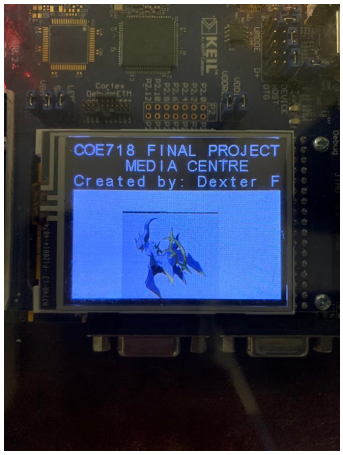

Fig. 4. The third photo in the gallery, an image of the Pokémon, Arceus.

C. MP3 Player

From a results perspective, the **MP3 Player** feature demonstrates a successful implementation of audio playback and user interaction, providing an engaging and functional media experience. The `mp3_music_player()` function effectively initializes the feature by setting up the display using `LCD_Template()`. This ensures a clean and clear interface, displaying relevant information such as ”MUSIC STREAM-

ING!" and "NOW PLAYING: VERDANTURF TOWN." These text outputs are rendered precisely, providing users with clear feedback on the current playback status.

The feature also incorporates visual elements, including a Pokeball and Ash & Pikachu, which are displayed alongside the track information. These graphics are handled by the `DisplayPokeball()` and `DisplayAsh()` functions and add an engaging visual component that enhances the user experience. The integration of graphical and audio components demonstrates the versatility of the media center.

Audio playback is achieved using the `stream_music()` function, which streams the Verdanturf Town theme directly from the LPC1768 microcontroller. The playback is smooth and uninterrupted, confirming the robustness of the audio streaming implementation. Volume control, facilitated through the potentiometer, works effectively, allowing users to adjust the playback volume in real-time, which adds a layer of interactivity and usability to the feature.

Overall, the MP3 Player achieves its intended goals, providing a visually appealing and fully functional audio playback feature. The results validate the seamless integration of hardware and software components, delivering a cohesive user experience. This feature stands out as an example of the successful application of embedded system design principles, effectively leveraging the capabilities of the MCB1700 evaluation board.

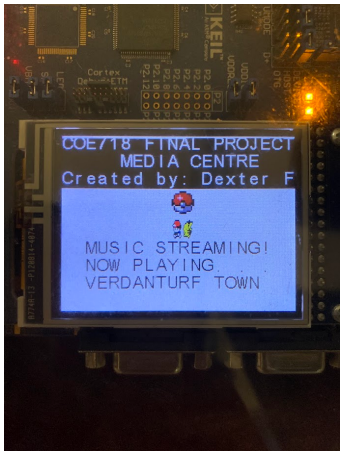

Fig. 5. The MP3 player menu.

D. Game: Snake

From a results perspective, the **Snake Game** is a standout feature of the media center, demonstrating both technical complexity and user engagement. The game initializes with a pre-game screen, offering clear instructions and rules such as "AVOID BORDERS!" and "THINK FAST!" This introductory screen provides users with essential guidance, ensuring they are prepared before the game begins. Once started, the game delivers a smooth and responsive experience, with the snake's movement controlled by the joystick and updated dynamically using the `update_snake()` function.

The gameplay mechanics, such as food generation and collision detection, are implemented effectively. The `generate_food()` function ensures that food appears in random, valid positions on the board while avoiding overlap with the snake's body. The visual variety of the food, toggling between "@" and "#" symbols, adds an engaging touch. Collision detection is handled through the `detect_collisions()` function, which determines whether the snake has collided with itself or the screen boundaries. This logic ensures a clear win/lose condition, creating a structured gameplay loop.

The dynamic rendering of the snake is another highlight of the implementation. The snake's appearance is visually distinct, with a square head, circular tail, and linked body segments, all updated in real-time as the game progresses. The screen setup, managed by `setup_window()`, clearly defines the play area with borders, providing a polished and professional look. The game's responsiveness and real-time updates demonstrate the efficient handling of user inputs and rendering on the MCB1700 evaluation board.

Upon game termination, the post-game screen displays the user's score and offers options to either restart or return to the Game Center. This thoughtful feature provides a seamless transition and enhances the user experience. Overall, the Snake Game successfully combines robust gameplay logic, responsive controls, and engaging visuals, showcasing the capabilities of the hardware and the effectiveness of the software design. It stands as a testament to the versatility and interactivity that can be achieved with embedded systems.

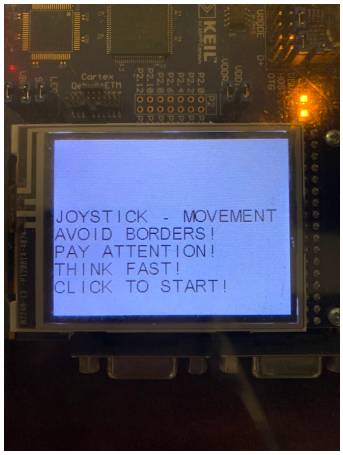

Fig. 6. Prompt Screen containing brief user instructions.

The **Snake Game** includes a particularly challenging aspect in the form of ASCII animations for the snake's different segments. Each part of the snake is represented with distinct ASCII characters: the head uses square brackets (`[]`), the body links use parentheses (`()`), and the tail uses a circular character (`o`). This differentiation adds a layer of complexity to the rendering process, as each segment requires unique logic to ensure the correct symbol is displayed and properly aligned with the rest of the snake. Managing these animations

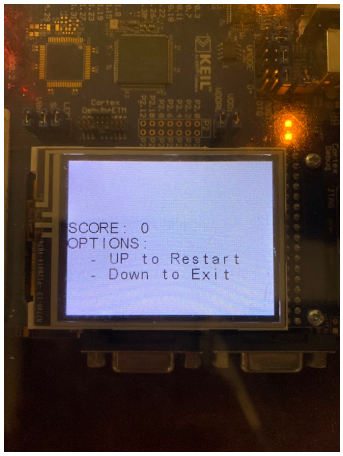

Fig. 7. Score menu when the user ends the game, either by touching the red borders of the screen or itself

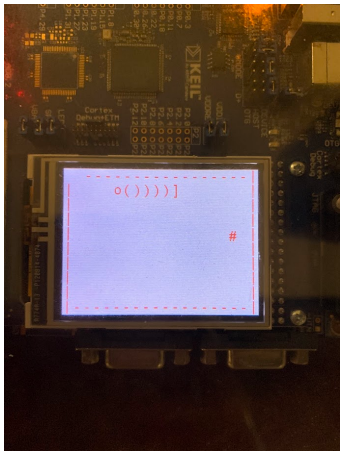

Fig. 8. Image of actual gameplay, where food generated is shown using ASCII keys.

dynamically as the snake moves and grows increases the computational load, particularly in ensuring the head and tail are updated seamlessly while maintaining the body links' consistency. Misalignments or overlaps could easily disrupt the visual clarity, but the implementation handles these updates effectively, showcasing meticulous attention to detail.

Another notable challenge encountered during the development of the game is related to the **food generation logic**. While the `generate_food()` function is designed to ensure food appears in valid locations that do not overlap with the snake's body, there are occasional instances where the food fails to load on the screen. This issue is likely due to the randomized generation process encountering conflicts with the snake's current position, especially as the snake grows larger and occupies more space. Additionally, there are sometimes noticeable delays in the appearance of new food items after one has been consumed. These delays, while not frequent, can disrupt the flow of the game and may stem from the computational time required to validate food positions or from the logic that toggles between the two symbols (`@` and `#`) used to represent the food.

These challenges highlight the inherent difficulties in combining real-time input handling, dynamic animations, and responsive gameplay in an embedded system. Despite these issues, the game effectively delivers a playable and engaging experience, and addressing these minor glitches could further enhance its robustness. The ASCII animations and food-generation logic demonstrate the intricate balancing act of managing system constraints while striving for an interactive and visually distinct gaming experience.

VII. Conclusion

In conclusion, this project successfully demonstrated the use of Hardware C programming on the ARM Cortex M3 chip using Keil uVision Software. All aspects of the "Snake" game, including smooth snake movement, responsive controls using ASCII characters, and dynamic food spawning, were implemented and function as intended. The game provides an engaging and interactive experience for the user, showcasing the capabilities of embedded systems for entertainment purposes.

The most challenging aspect of the project was animating the snake using ASCII keys, which required precise timing and synchronization between user input and game logic. Additionally, addressing occasional delays in the appearance of the "food" presented another significant difficulty.

Overall, this project illustrates how embedded systems can be leveraged to create entertaining and functional applications using simple hardware and logic-based design principles.

A. References

References

[1] G. N. Khan, "Final Project: Media Center," Toronto Metropolitan University, Toronto, 2023.
[2] ASCII Code, "The extended ASCII table," Accessed: Nov. 30, 2024. [Online]. Available: https://www.ascii-code.com/
[3] Keil, "µVision IDE User's Guide," Accessed: Nov. 30, 2024. [Online]. Available: https://www.keil.com/support/man/docs/uv4/default.htm

Acknowledgment

I would like to acknowledge my COE718 Embedded Systems Design Teaching Assistant, Mr. David Wu, for his kindness and patience throughout the Fall 2024 semester in dealing with me as his student.

APPENDIX: Code

```
/*----------------------------------------------------------------------------
 * CMSIS-RTOS 'main' function template
 *---------------------------------------------------------------------------*/

#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>

#include "Board_Buttons.h"
#include "GLCD.h"
#include "KBD.h"
#include "LED.h"
#include "LPC17xx.h"
#include "AUDIO/usbdmain.h"
#include "photo_gallery.h"
#include "AUDIO/usb.h"
#include "AUDIO/usbcfg.h"
#include "AUDIO/usbhw.h"
#include "AUDIO/usbcore.h"
#include "AUDIO/usbaudio.h"
#include "cmsis_os.h"
#include "snake.h"

// Font index 16x24
#define __FI 1

// Joystick Values

#define UP 8
#define DOWN 32
#define LEFT 64
#define RIGHT 16
#define NEUTRAL 0
#define PRESSED 1

// ARRAY SIZES
#define PHOTO_GALLERY_ARRAY_SIZE 4
#define MENU_ARRAY_SIZE 3
#define GAME_ARRAY_SIZE 1

// STATES
#define PHOTO_GALLERY 0
```

```
#define MP3_PLAYER 1
#define GAME_CENTER 2
#define HOME_SCREEN 3

#define SNAKE 4

// Global Variables used for flow control between states
int selector = 0;
int prev_direction, direction;


void LCD_Template() {
    // Clear graphical LCD display
    GLCD_Clear(White);

    // Set a blue background with yellow text fot the LCD display
    // Add the Headers from the lab manual

    GLCD_SetBackColor(Black);
    GLCD_SetTextColor(White);
    GLCD_DisplayString(0, 0, __FI, (unsigned char *)"COE718 FINAL PROJECT");
            GLCD_DisplayString(1, 0, __FI, (unsigned char *)"     MEDIA CENTRE         ");
    GLCD_SetTextColor(White);
            GLCD_DisplayString(2, 0, __FI, (unsigned char *)"Created by: Dexter F.");
    GLCD_SetBackColor(White);
    GLCD_SetTextColor(Black);
}

int menu_selector(int selector, int joystick_value, int array_size) {
    if (joystick_value == UP) {
        if (selector == 0) {
            selector = array_size - 1;
        } else {
            selector--;
        }

    } else if (joystick_value == DOWN) {
        if (selector == array_size - 1) {
            selector = 0;
        } else {
            selector++;
        }
    }
    return selector;
```

```
}

int side_ways_selector(int selector, int joystick_value, int array_size) {
    if (joystick_value == LEFT) {
        if (selector == 0) {
            selector = array_size - 1;
        } else {
            selector--;
        }

    } else if (joystick_value == RIGHT) {
        if (selector == array_size - 1) {
            selector = 0;
        } else {
            selector++;
        }
    }
    return selector;
}
void screen_options_select(int selector, char *screen_options[],
                           int size_of_array) {
    int i;
    for (i = 0; i < size_of_array; i++) {
        if (i == selector) {
            GLCD_SetBackColor(Green);
        } else {
            GLCD_SetBackColor(White);
        }
        GLCD_DisplayString(6 + i, 0, __FI, (unsigned char *)screen_options[i]);
    }
}

int home_screen_main(int previous_page, char *home_screen_options[]) {
    if (previous_page != HOME_SCREEN) {
        LCD_Template();
        selector = 0;
    }
    direction = get_button();
    if (prev_direction == NEUTRAL) {
        selector = menu_selector(selector, direction, MENU_ARRAY_SIZE);
        screen_options_select(selector, home_screen_options, MENU_ARRAY_SIZE);
    }
    prev_direction = direction;
```

```
    // button
    if (get_button() == PRESSED) {
        // Selector's value is equal to the page's number
        return selector;
    }
    return HOME_SCREEN;
}

int photo_gallery(int previous_page) {
    if (previous_page != PHOTO_GALLERY) {
        LCD_Template();
    }
    direction = get_button();
    if (prev_direction == NEUTRAL) {
        selector =
            side_ways_selector(selector, direction, PHOTO_GALLERY_ARRAY_SIZE);
        GLCD_Bitmap(75, 100, 150, 150,
                    (unsigned char *)images_for_photo_gallery[selector]);
    }
    prev_direction = direction;

    if (Buttons_GetState()) return HOME_SCREEN;
    return PHOTO_GALLERY;
}

const uint16_t pokeball_image_bitmap[32][32] = { // Pokeball
    { 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF,
0xFFFF, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000,
0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF,
0xFFFF },
    { 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF,
0xFFFF, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000,
0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF,
0xFFFF },
    { 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF,
0xFFFF, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000,
0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF,
0xFFFF },
    { 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000,
0x0000, 0xAD55, 0xAD55, 0xAD55, 0xAD55, 0xAD55, 0xAD55, 0xAD55, 0xAD55, 0xAD55,
0xAD55, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0xFFFF, 0xFFFF, 0xFFFF,
0xFFFF, 0xFFFF },
    { 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000,
0x0000, 0xAD55, 0xAD55, 0xAD55, 0xAD55, 0xAD55, 0xAD55, 0xAD55, 0xAD55, 0xAD55,
```

0xAD55, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF },
{ 0xFFFF, 0xFFFF, 0xFFFF, 0x0000, 0x0000, 0xAD55, 0xAD55, 0xAD55, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xAD55, 0xAD55, 0xAD55, 0xAD55, 0xAD55, 0xAD55, 0xAD55, 0xAD55, 0xAD55, 0xAD55, 0xAD55, 0x0000, 0x0000, 0xFFFF, 0xFFFF, 0xFFFF },
{ 0xFFFF, 0xFFFF, 0xFFFF, 0x0000, 0x0000, 0xAD55, 0xAD55, 0xAD55, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xAD55, 0xAD55, 0xAD55, 0xAD55, 0xAD55, 0xAD55, 0xAD55, 0xAD55, 0xAD55, 0xAD55, 0xAD55, 0x0000, 0x0000, 0xFFFF, 0xFFFF, 0xFFFF },
{ 0xFFFF, 0xFFFF, 0xFFFF, 0x0000, 0x0000, 0xAD55, 0xAD55, 0xAD55, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xAD55, 0xAD55, 0xAD55, 0xAD55, 0xAD55, 0xAD55, 0xAD55, 0xAD55, 0xAD55, 0xAD55, 0xAD55, 0x0000, 0x0000, 0xFFFF, 0xFFFF, 0xFFFF },
{ 0xFFFF, 0xFFFF, 0xFFFF, 0x0000, 0x0000, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0xAD55, 0xAD55, 0xAD55, 0xAD55, 0xAD55, 0xAD55, 0xAD55, 0xAD55, 0x0000, 0x0000, 0xFFFF, 0xFFFF, 0xFFFF },
{ 0xFFFF, 0xFFFF, 0xFFFF, 0x0000, 0x0000, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0xAD55, 0xAD55, 0xAD55, 0xAD55, 0xAD55, 0xAD55, 0xAD55, 0xAD55, 0x0000, 0x0000, 0xFFFF, 0xFFFF, 0xFFFF },
{ 0xFFFF, 0xFFFF, 0xFFFF, 0x0000, 0x0000, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0xAD55, 0xAD55, 0xAD55, 0xAD55, 0xAD55, 0xAD55, 0xAD55, 0xAD55, 0x0000, 0x0000, 0xFFFF, 0xFFFF, 0xFFFF },
{ 0x0000, 0x0000, 0x0000, 0xFFFF, 0xFFFF, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0xAD55, 0xAD55, 0xAD55, 0xAD55, 0xAD55, 0xAD55, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0xAD55, 0xAD55, 0x0000, 0x0000, 0x0000 },
{ 0x0000, 0x0000, 0x0000, 0xFFFF, 0xFFFF, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0xAD55, 0xAD55, 0xAD55, 0xAD55, 0xAD55, 0xAD55, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0xAD55, 0xAD55, 0x0000, 0x0000, 0x0000 },
{ 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0xD020, 0xD020, 0xD020, 0xD020, 0xD020, 0xD020, 0x0000, 0x0000, 0xFFFF, 0xFFFF, 0xFFFF, 0xAD55, 0xAD55, 0xAD55, 0x0000, 0x0000, 0xB820, 0xB820, 0xB820, 0x9020, 0x9020, 0x9020, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000 },
{ 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0xD020, 0xD020, 0xD020, 0xD020, 0xD020, 0xD020, 0x0000, 0x0000, 0xFFFF, 0xFFFF, 0xFFFF, 0xAD55, 0xAD55, 0xAD55, 0x0000, 0x0000, 0xB820, 0xB820, 0xB820, 0x9020, 0x9020, 0x9020, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000 },
{ 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0xD020, 0xD020, 0xD020, 0xD020, 0xD020, 0xD020, 0x0000, 0x0000, 0xFFFF, 0xFFFF, 0xFFFF, 0xAD55, 0xAD55, 0xAD55, 0x0000,

0x0000, 0xB820, 0xB820, 0xB820, 0x9020, 0x9020, 0x9020, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000 },
{ 0x0000, 0x0000, 0x0000, 0xD020, 0xD020, 0xD020, 0xD020, 0xD020, 0xD020, 0xD020, 0xD020, 0xD020, 0xD020, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0xB820, 0xB820, 0xB820, 0xB820, 0xB820, 0xB820, 0xB820, 0xB820, 0x9020, 0x9020, 0x0000, 0x0000, 0x0000 },
{ 0x0000, 0x0000, 0x0000, 0xD020, 0xD020, 0xD020, 0xD020, 0xD020, 0xD020, 0xD020, 0xD020, 0xD020, 0xD020, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0xB820, 0xB820, 0xB820, 0xB820, 0xB820, 0xB820, 0xB820, 0xB820, 0x9020, 0x9020, 0x0000, 0x0000, 0x0000 },
{ 0x0000, 0x0000, 0x0000, 0xD020, 0xD020, 0xD020, 0xD020, 0xD020, 0xD020, 0xD020, 0xD020, 0xD020, 0xD020, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0xB820, 0xB820, 0xB820, 0xB820, 0xB820, 0xB820, 0xB820, 0xB820, 0x9020, 0x9020, 0x0000, 0x0000, 0x0000 },
{ 0x0000, 0x0000, 0x0000, 0xB820, 0xB820, 0xD020, 0xD020, 0xD020, 0xD020, 0xD020, 0xD020, 0xFCD2, 0xFCD2, 0xD020, 0xD020, 0xD020, 0xD020, 0xD020, 0xD020, 0xD020, 0xD020, 0xB820, 0xB820, 0xB820, 0xB820, 0xB820, 0xB820, 0x9020, 0x9020, 0x0000, 0x0000, 0x0000 },
{ 0x0000, 0x0000, 0x0000, 0xB820, 0xB820, 0xD020, 0xD020, 0xD020, 0xD020, 0xD020, 0xD020, 0xFCD2, 0xFCD2, 0xD020, 0xD020, 0xD020, 0xD020, 0xD020, 0xD020, 0xD020, 0xD020, 0xB820, 0xB820, 0xB820, 0xB820, 0xB820, 0xB820, 0x9020, 0x9020, 0x0000, 0x0000, 0x0000 },
{ 0xFFFF, 0xFFFF, 0xFFFF, 0x0000, 0x0000, 0xD020, 0xD020, 0xD020, 0xFCD2, 0xFCD2, 0xFCD2, 0xFFFF, 0xFFFF, 0xFCD2, 0xFCD2, 0xFCD2, 0xD020, 0xD020, 0xD020, 0xD020, 0xD020, 0xB820, 0xB820, 0xB820, 0xB820, 0xB820, 0xB820, 0x0000, 0x0000, 0xFFFF, 0xFFFF, 0xFFFF },
{ 0xFFFF, 0xFFFF, 0xFFFF, 0x0000, 0x0000, 0xD020, 0xD020, 0xD020, 0xFCD2, 0xFCD2, 0xFCD2, 0xFFFF, 0xFFFF, 0xFCD2, 0xFCD2, 0xFCD2, 0xD020, 0xD020, 0xD020, 0xD020, 0xD020, 0xB820, 0xB820, 0xB820, 0xB820, 0xB820, 0xB820, 0x0000, 0x0000, 0xFFFF, 0xFFFF, 0xFFFF },
{ 0xFFFF, 0xFFFF, 0xFFFF, 0x0000, 0x0000, 0xD020, 0xD020, 0xD020, 0xFCD2, 0xFCD2, 0xFCD2, 0xFFFF, 0xFFFF, 0xFCD2, 0xFCD2, 0xFCD2, 0xD020, 0xD020, 0xD020, 0xD020, 0xD020, 0xB820, 0xB820, 0xB820, 0xB820, 0xB820, 0xB820, 0x0000, 0x0000, 0xFFFF, 0xFFFF, 0xFFFF },
{ 0xFFFF, 0xFFFF, 0xFFFF, 0x0000, 0x0000, 0xB820, 0xB820, 0xB820, 0xD020, 0xD020, 0xD020, 0xFCD2, 0xFCD2, 0xD020, 0xD020, 0xD020, 0xD020, 0xD020, 0xD020, 0xB820, 0xB820, 0xB820, 0xB820, 0xB820, 0x9020, 0x9020, 0x9020, 0x0000, 0x0000, 0xFFFF, 0xFFFF, 0xFFFF },
{ 0xFFFF, 0xFFFF, 0xFFFF, 0x0000, 0x0000, 0xB820, 0xB820, 0xB820, 0xD020, 0xD020, 0xD020, 0xFCD2, 0xFCD2, 0xD020, 0xD020, 0xD020, 0xD020, 0xD020, 0xD020, 0xB820, 0xB820, 0xB820, 0xB820, 0xB820, 0x9020, 0x9020, 0x9020, 0x0000, 0x0000, 0xFFFF, 0xFFFF, 0xFFFF },
{ 0xFFFF, 0xFFFF, 0xFFFF, 0x0000, 0x0000, 0xB820, 0xB820, 0xB820, 0xD020, 0xD020, 0xD020, 0xFCD2, 0xFCD2, 0xD020, 0xD020, 0xD020, 0xD020, 0xD020, 0xD020, 0xB820,

```
0xB820, 0xB820, 0xB820, 0xB820, 0x9020, 0x9020, 0x9020, 0x0000, 0x0000, 0xFFFF,
0xFFFF, 0xFFFF },
    { 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000,
0x0000, 0xB820, 0xB820, 0xB820, 0xB820, 0xB820, 0xB820, 0xB820, 0xB820, 0xB820,
0xB820, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF,
0xFFFF },
    { 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000,
0x0000, 0xB820, 0xB820, 0xB820, 0xB820, 0xB820, 0xB820, 0xB820, 0xB820, 0xB820,
0xB820, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF,
0xFFFF },
    { 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF,
0xFFFF, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000,
0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF,
0xFFFF },
    { 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF,
0xFFFF, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000,
0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF,
0xFFFF },
    { 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF,
0xFFFF, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000,
0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF,
0xFFFF }
                        };

// Function to display the flipped bitmap on the screen
void DisplayPokeball() { // Pokeball
    // Display the pokeball
    GLCD_Bitmap(150, 75, 32, 32, (unsigned char *)pokeball_image_bitmap);
}
const uint16_t Ash_image_bitmap[32][32] = {
    { 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF,
0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF,
0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF,
0xFFFF, 0xFFFF },
    { 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF,
0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF,
0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF,
0xFFFF, 0xFFFF },
    { 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF,
0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF,
0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF,
0xFFFF, 0xFFFF },
    { 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0000, 0x0000, 0x0000, 0xFFFF, 0xFFFF, 0xFFFF,
0xFFFF, 0x0000, 0x0000, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF,
```

0xFFFF, 0xFFFF, 0xFFFF, 0x0000, 0x0000, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF,
0xFFFF, 0xFFFF },
{ 0xFFFF, 0xFFFF, 0xFFFF, 0xFFDF, 0x0841, 0x0000, 0x0000, 0xFFFF, 0xFFFF, 0xFFFF,
0xFFFF, 0x0000, 0x0000, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFDF, 0xFFFF,
0xFFFF, 0xFFFF, 0xFFFF, 0x0841, 0x0000, 0xFFFF, 0xF7BE, 0xFFFF, 0xFFFF, 0xFFFF,
0xFFFF, 0xFFFF },
{ 0xFFFF, 0xFFFF, 0xFFFF, 0x0000, 0xFFFF, 0xFFFF, 0x0000, 0x0000, 0x0000, 0x0000,
0x0000, 0xFFFF, 0xFFFF, 0x0000, 0xFFFF, 0xFFFF, 0xFFFF, 0x0000, 0x0000, 0x0001, 0x0000,
0x0000, 0x0000, 0xFFE0, 0xF7E0, 0x0020, 0xFFFE, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF,
0xFFFF },
{ 0xFFFF, 0xFFFF, 0xFFFF, 0x0000, 0xFFFF, 0xFFFF, 0x0000, 0x0000, 0x0000, 0x0000,
0x0000, 0xFFFF, 0xFFFF, 0x0000, 0xFFFF, 0xFFFF, 0xFFDE, 0x0000, 0x0000, 0x0840,
0x0000, 0x0000, 0x0000, 0xFFE5, 0xF7E3, 0x0000, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF,
0xFFFF, 0xFFFF },
{ 0xFFFF, 0xFFFF, 0x0000, 0xFFFF, 0xFFFF, 0x0000, 0xFFFF, 0xFFFF, 0x0000, 0xFFFF,
0xFFFF, 0x0000, 0xFFFF, 0x0000, 0xFFFF, 0xFFFF, 0x0001, 0xFFE0, 0xEFC8, 0xEFC8,
0xFFE0, 0xFFE0, 0xFFE0, 0x0000, 0x0000, 0xFFC0, 0x0000, 0xFFFF, 0xFFFF, 0xFFFF,
0xFFFF, 0xFFFF },
{ 0xFFFF, 0xFFFF, 0xFFFF, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0xFFFF,
0xFFFF, 0x0000, 0x0000, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0020, 0x0820, 0xFFE1,
0xFFE0, 0xFFE0, 0xFFE0, 0xF7C3, 0x0000, 0x0020, 0x0000, 0xFFFF, 0xFFFF, 0xFFFF,
0xFFFF, 0xFFFF },
{ 0xFFFF, 0xFFFF, 0xFFFF, 0x0020, 0x0841, 0x0000, 0x0000, 0x0000, 0x0000, 0xFFFF,
0xFFFF, 0x0000, 0x0000, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0821, 0x0800, 0xFFE0,
0xFFE1, 0xFFE0, 0xF7E0, 0xFFE0, 0x0000, 0x0000, 0x0000, 0xF7DF, 0xFFFF, 0xFFFF,
0xFFFF, 0xFFFF },
{ 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000,
0x0000, 0x049F, 0x0000, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0000, 0xF7C6, 0x0001,
0xF7C7, 0xFFE0, 0x61E0, 0x61C0, 0x61C0, 0x0000, 0x69C0, 0x0000, 0xFFFF, 0xFFFF,
0xFFFF, 0xFFFF },
{ 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000,
0x0000, 0x049F, 0x0000, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0000, 0xF7C6, 0x0000,
0xFFE8, 0xFFE0, 0x6200, 0x61E0, 0x61E0, 0x0001, 0x61E0, 0x0800, 0xFFFF, 0xFFFF,
0xFFFF, 0xFFFF },
{ 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0000, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0000,
0x0000, 0x0C7E, 0x0000, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0000, 0xF7C6, 0xEFC8,
0xFFE0, 0xFFE0, 0xFFE0, 0xFFE0, 0xFFE0, 0x0020, 0x61E0, 0x69C0, 0x0000, 0xFFFF,
0xFFFF, 0xFFFF },
{ 0xFFFF, 0xFFFF, 0xFFFF, 0x0000, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF,
0xFFFF, 0x0022, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0800, 0xFFE1, 0xFFE1,
0xFFE0, 0xF7E0, 0x6200, 0x61E0, 0x61E0, 0x0000, 0x61C0, 0x0800, 0xFFFF, 0xFFFF,
0xFFFF, 0xFFFF },
{ 0xFFFF, 0xFFFF, 0xFFFF, 0x0000, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF,
0xFFFF, 0x0000, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0820, 0xFFE0, 0xFFE0,

0xFFE0, 0xFFC1, 0x5980, 0x69C0, 0x71E2, 0x0000, 0x6A00, 0x0000, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF },
{ 0xFFFF, 0xFFFF, 0xFFFF, 0x0000, 0xFFFF, 0x0000, 0xFFFF, 0xFFFF, 0x0000, 0xFFFF, 0xFFFF, 0x0000, 0x0000, 0xFFFF, 0xFFFF, 0xFFFF, 0x0000, 0xFFE0, 0xFFE0, 0xFFE0, 0xD8E0, 0xF7E0, 0xFFE0, 0xFFE0, 0x0000, 0x0000, 0xFFE0, 0xFFE0, 0x0000, 0xFFFF, 0xFFFF, 0xFFFF },
{ 0xFFFF, 0xFFFF, 0xFFFF, 0x0000, 0xFFFF, 0x0000, 0xFFFF, 0xFFFF, 0x0000, 0xFFFF, 0xFFFF, 0x0000, 0x0000, 0xFFFF, 0xFFFF, 0xFFFF, 0x0000, 0xFFE0, 0xF7C0, 0xFFE1, 0xC8C0, 0xF7A1, 0xFFE0, 0xFFE0, 0x0000, 0x0000, 0xF7E0, 0xF7E0, 0x0000, 0xFFFF, 0xFFFF, 0xFFFF },
{ 0xFFFF, 0xFFFF, 0xFFFF, 0x0000, 0xFFFF, 0x0000, 0xFFFF, 0xFFFF, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0xFFFF, 0xFFFF, 0x0001, 0xFFE0, 0xF7C6, 0xEFC8, 0xFFE0, 0xFFE0, 0xFFE0, 0xFFE0, 0x0000, 0xFFE0, 0x0000, 0x0000, 0x0000, 0xFFFF, 0xFFFF, 0xFFFF },
{ 0xFFFF, 0xFFFF, 0x0000, 0x0000, 0xB8C4, 0xB8E3, 0xC0C3, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0xFFFF, 0xFFFF, 0xFFBF, 0x1060, 0xFFE4, 0xFFE4, 0xFFE0, 0xFFE0, 0xFFE0, 0x0840, 0xEFA8, 0xFFE0, 0x0000, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF },
{ 0xFFFF, 0xFFFF, 0x0000, 0x0000, 0xF860, 0xE820, 0xF020, 0x0000, 0x0000, 0x0000, 0x0800, 0x0000, 0x0000, 0x0000, 0xFFFF, 0xFFFF, 0xFFFF, 0x0000, 0xFFE0, 0xFFE0, 0xFFE0, 0xFFE0, 0xFFE0, 0x0000, 0xFFE3, 0xFFE1, 0x0040, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF },
{ 0xFFFF, 0x0000, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xF800, 0xF800, 0xF800, 0xF800, 0xF800, 0x0000, 0x0000, 0x0000, 0xFFFF, 0xFFFF, 0xFFFF, 0x0000, 0xFFE0, 0xFFE0, 0xFFE0, 0xFFE0, 0xFFE0, 0x0000, 0x0000, 0xFFC0, 0xFFE0, 0x0000, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF },
{ 0xFFFF, 0x0000, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFE, 0xF800, 0xF800, 0xF800, 0xF800, 0xF800, 0x0020, 0x0000, 0x0000, 0xFFFF, 0xFFFF, 0xFFDF, 0x0000, 0xFFE2, 0xFFE0, 0xFFE0, 0xFFE0, 0xFFE0, 0x0000, 0x0000, 0xFFC0, 0xFFE0, 0x0000, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF },
{ 0xFFFF, 0xFFFF, 0x0000, 0x0000, 0xF7FF, 0xF800, 0xF800, 0xF800, 0xF800, 0xF800, 0xF800, 0xF800, 0x0000, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0820, 0xF7E0, 0xF7E0, 0xFFE1, 0xFFE0, 0xFFE0, 0x0001, 0xFFE0, 0xFFE1, 0xFFE0, 0x0000, 0xFFFF, 0xFFFF, 0xFFFF },
{ 0xFFFF, 0xFFFF, 0xFFFF, 0x0000, 0xB8A4, 0xF800, 0xF800, 0xF800, 0xF800, 0xF800, 0xF800, 0xF800, 0xF800, 0x0000, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFDF, 0xFFFF, 0x0840, 0x0840, 0xF7A2, 0xFFE0, 0x0840, 0x0000, 0x0840, 0xFFC0, 0xFFE0, 0xEFA9, 0x0000, 0xFFFF, 0xFFFF },
{ 0xFFFF, 0xFFFF, 0xFFFF, 0x0000, 0xE800, 0xF800, 0xF800, 0xF800, 0xF800, 0xF800, 0xF800, 0xF020, 0x0000, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0000, 0x0000, 0xEFE2, 0xFFE0, 0x0820, 0x0000, 0x0000, 0xFFC0, 0xFFE0, 0xFFC2, 0x0000, 0xFFFF, 0xFFFF },
{ 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0000, 0xF800, 0xF800, 0xF800, 0xF800, 0xF800, 0xF800, 0x0000, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF,

```
0xFFFF, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0xFFFF,
0xFFFF },
   { 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0020, 0xF800, 0xF800, 0xF800, 0xF800, 0xF800,
0xF800, 0x0000, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF,
0xFFFF, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0020, 0x0000, 0x0000, 0x0000, 0xFFFF,
0xFFFF },
   { 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000,
0x0000, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF,
0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0000, 0x0000, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF,
0xFFFF, 0xFFFF },
   { 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000,
0x0000, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF,
0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0000, 0x0000, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF,
0xFFFF, 0xFFFF },
   { 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF,
0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF,
0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF,
0xFFFF, 0xFFFF },
   { 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF,
0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF,
0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF,
0xFFFF, 0xFFFF },
   { 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF,
0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF,
0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF,
0xFFFF, 0xFFFF }
};
void DisplayAsh() { // Pokeball
   // Display the ash and pikachu
   GLCD_Bitmap(150, 112, 32, 32, (unsigned char *)Ash_image_bitmap);
}
int mp3_music_player() {
   LCD_Template();
   GLCD_DisplayString(6, 2, __FI, (unsigned char *)"MUSIC STREAMING!");
           GLCD_DisplayString(7, 2, __FI, (unsigned char *)"NOW PLAYING. . . ");
           GLCD_DisplayString(8, 2, __FI, (unsigned char *) "VERDANTURF TOWN");
   DisplayPokeball();
                   DisplayAsh();
           stream_music();
   return HOME_SCREEN;
}

int game_center(int previous_page) {
   if (previous_page != GAME_CENTER) {
```

```
        LCD_Template(); // Clear the screen and set up the game environment
    }
    // Skip menu selection and joystick handling; go straight into the game
    return SNAKE; // Immediately start the snake game
}


void pre_game_screen() {
    GLCD_Clear(White);
    GLCD_SetBackColor(White);
    GLCD_SetTextColor(Black);
    GLCD_DisplayString(4, 0, __FI, (unsigned char *)"JOYSTICK - MOVEMENT");
    GLCD_DisplayString(5, 0, __FI, (unsigned char *)"AVOID BORDERS!");
    GLCD_DisplayString(6, 0, __FI, (unsigned char *)"PAY ATTENTION!");
    GLCD_DisplayString(7, 0, __FI, (unsigned char *)"THINK FAST!");
    GLCD_DisplayString(8, 0, __FI, (unsigned char *)"CLICK TO START!");
    GLCD_SetTextColor(Blue);
}

int post_game_screen(int score, int game) {
    char score_str[5];
    snprintf(score_str, 5, "%d", score);
    GLCD_Clear(White);
    GLCD_SetBackColor(White);
    GLCD_SetTextColor(Black);
    GLCD_DisplayString(4, 0, __FI, (unsigned char *)"SCORE:");
    GLCD_DisplayString(4, 7, __FI, (unsigned char *)score_str);
    GLCD_DisplayString(5, 0, __FI, (unsigned char *)"OPTIONS:");
    GLCD_DisplayString(6, 0, __FI, (unsigned char *)"  - UP -> Restart");
    GLCD_DisplayString(7, 0, __FI, (unsigned char *)"  - DOWN -> Exit");
    GLCD_SetTextColor(Blue);
    while (1) {
        if (get_button() == UP) {
            return game;
        } else if (get_button() == DOWN) {
            return GAME_CENTER;
        }
    }
}
int snake_main(int previous_page) {
    int game_score;

    if (previous_page != SNAKE) {
        pre_game_screen();
```

```
    }
    while (!get_button());
    GLCD_Clear(White);
    GLCD_SetBackColor(White);
    game_score = snake_game();
    return post_game_screen(game_score, SNAKE);
}

int main(void) {
    int page_to_show = HOME_SCREEN;
    int new_page = HOME_SCREEN;
    int previous_page = HOME_SCREEN;

    char *home_screen_options[] = {"Photo Gallery", "MP3 Player", "Game Center"};
    char *game_center_options[] = {"ASCII SNAKE GAME"};
    LED_Init();
    GLCD_Init();
    LCD_Template();
    KBD_Init();
    Buttons_Initialize();

    for (;;) {
        previous_page = page_to_show;  // Update previous_page before
                                       // page_to_show is updated
        page_to_show = new_page;
        switch (page_to_show) {
            case HOME_SCREEN:
                new_page = home_screen_main(previous_page, home_screen_options);
                break;
            case PHOTO_GALLERY:
                new_page = photo_gallery(previous_page);
                break;
            case MP3_PLAYER:
                // always returns to home screen
                new_page = mp3_music_player();
                break;
            case GAME_CENTER:
                new_page = game_center(previous_page);
                break;
                                                    case SNAKE:
                                                                    new_page =
snake_main(previous_page);
                                                                    break;
        }
```

```
    }
}


#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>

#include "Board_Buttons.h"
#include "GLCD.h"
#include "KBD.h"

// Direction macros
#define UP 8
#define DOWN 32
#define LEFT 64
#define RIGHT 16
#define NEUTRAL 0
#define PRESSED 1

#define SCREEN_LENGTH 9
#define SCREEN_WIDTH 19

// Font index 16x24
#define __FI 1

int snake[144][2];
int food_x, food_y;

int snake_x, snake_y;
int snake_length = 3;
int food_toggle = 0; // Global variable to alternate between '@' and '#'

void delay() {
    int i;
    for (i = 0; i < 2500000; i++) {
        __nop();
        __nop();
    }
}

void setup_window() {
    int i;
```

```
    // Set color for the borders (red)
    GLCD_SetTextColor(Red);

    GLCD_DisplayString(0, 0, __FI,
                    (unsigned char *)"------------------------------");
    GLCD_DisplayString(9, 0, __FI,
                    (unsigned char *)"------------------------------");
    for (i = 0; i <= SCREEN_LENGTH; i++) {
        GLCD_DisplayString(i, 0, __FI, (unsigned char *)"|");
        GLCD_DisplayString(i, 19, __FI, (unsigned char *)"|");
    }
}

void init_game() {
    GLCD_SetTextColor(Black);
    snake_x = 7;
    snake_y = 5;
    // Snake head at index 0
    snake[0][0] = snake_x;
    snake[0][1] = snake_y;

    snake[1][0] = snake_x;
    snake[1][1] = snake_y - 1;

    snake[2][0] = snake_x;
    snake[2][1] = snake_y - 2;
}

int generate_food_helper(int food_x, int food_y) {
    // Returns true if the food generated is not in the snake
    int i;
    for (i = 0; i < snake_length; i++) {
        if (food_x == snake[i][0] && food_y == snake[i][1]) {
            return 0;
        }
    }
    return 1;
}

void generate_food() {
    do {
        food_y = rand() % (SCREEN_LENGTH - 1) + 1;
        food_x = rand() % (SCREEN_WIDTH - 1) + 1;
```

```
    } while (!generate_food_helper(food_x, food_y));
}

int detect_collisions() {
    // Checks if the snake ate itself
    int i;
    for (i = 1; i < snake_length; i++)
        if (snake[0][0] == snake[i][0] && snake[0][1] == snake[i][1]) return 1;
    if (snake[0][0] == 0 || snake[0][0] == SCREEN_LENGTH || snake[0][1] == 0 ||
        snake[0][1] == SCREEN_WIDTH)
        return 1;

    return 0;
}

void update_snake() {
    int i;

    // Clear the tail of the snake
    GLCD_DisplayChar(snake[snake_length - 1][0], snake[snake_length - 1][1], 1, ' '); // Clear last
segment
    GLCD_DisplayChar(snake[snake_length - 1][0], snake[snake_length - 1][1] + 1, 1, ' '); // Clear
overflow for multi-char

    for (i = snake_length - 1; i > 0; i--) {
        snake[i][1] = snake[i - 1][1];
        snake[i][0] = snake[i - 1][0];
    }

    // Update the head position
    snake[0][0] = snake_x;
    snake[0][1] = snake_y;

    // Display the snake with square segments, body links, and circular nodes
    for (i = 0; i < snake_length; i++) {
        if (i == 0) {
            GLCD_DisplayChar(snake[i][0], snake[i][1], 1, '['); // Square head start
            GLCD_DisplayChar(snake[i][0], snake[i][1] + 1, 1, ']'); // Square head end
        } else if (i == snake_length - 1) {
            GLCD_DisplayChar(snake[i][0], snake[i][1], 1, 'o'); // Circular tail
        } else {
            GLCD_DisplayChar(snake[i][0], snake[i][1], 1, '('); // Body link start
            GLCD_DisplayChar(snake[i][0], snake[i][1] + 1, 1, ')'); // Body link end
        }
```

```
    }
}

int snake_game() {
    int direction = RIGHT; // Start moving right
    int score = 0;
    snake_length = 3;
    init_game();
    setup_window();
    generate_food();
    GLCD_DisplayString(food_y, food_x, __FI, (unsigned char *)"*");

    while (1) {
        if (get_button()) {
            direction = get_button();
        }

        switch (direction) {
            case UP:
                if (snake_x - 1 != snake[1][0]) snake_x = snake_x - 1;
                break;
            case DOWN:
                if (snake_x + 1 != snake[1][0]) snake_x = snake_x + 1;
                break;
            case LEFT:
                if (snake_y - 1 != snake[1][1]) snake_y = snake_y - 1;
                break;
            case RIGHT:
                if (snake_y + 1 != snake[1][1]) snake_y = snake_y + 1;
                break;
            default:
                break;
        }

        update_snake();
        if (snake[0][0] == food_y && snake[0][1] == food_x) {
            score += 1;
            snake_length++;
            generate_food();

            // Toggle between '@' and '#' for food display
            if (food_toggle == 0) {
                GLCD_DisplayString(food_y, food_x, __FI, (unsigned char *)"@");
                food_toggle = 1; // Set toggle for next food
```

```
            } else {
                GLCD_DisplayString(food_y, food_x, __FI, (unsigned char *)"#");
                food_toggle = 0; // Reset toggle for next food
            }
        }

        if (detect_collisions()) return score;
        delay();
    }
}
```