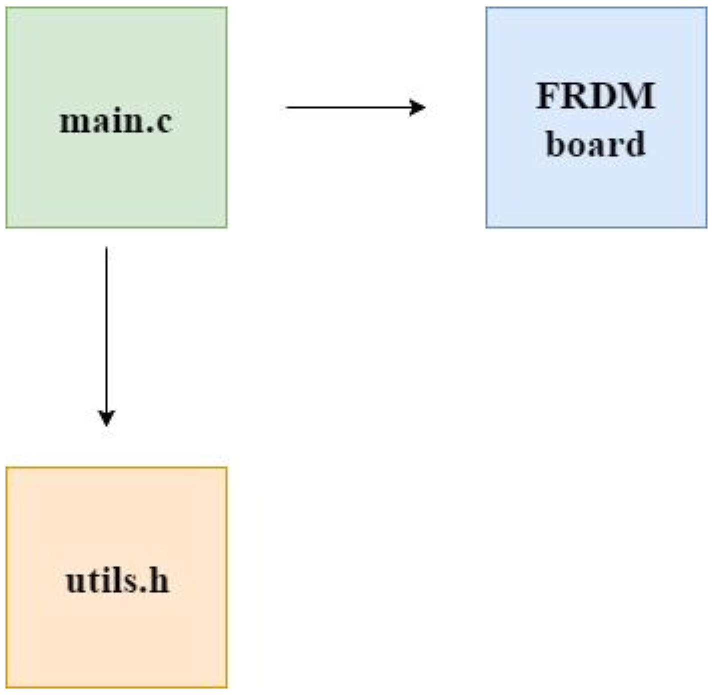
Rohit Curucundhi (rc563), Keshav Iyer (ki79)
CS 3420 Lab 6/Final Project

Introduction

Our project implements a maze game using the accelerometer on our board. A user can orient the board onto any of three axes to move forward in that direction. At the beginning of the game, users can provide input to determine whether they would like a random map, or a preset map that we have created. On a random map input, users can adjust settings for number of pieces and maximum gold. During the game, users can pick up gold along the way to track their progress and see how much of map they have explored. However, the goal is to find the finish in as short an amount of time as possible. So, more turns means a lower score! Upon moving off map, the board will light up with the red LED to indicate that a player has been blocked. A blue LED flash indicates the player has completed the game, and a green LED tells the player that they can continue on their current path, or have chosen a direction in which they can move. A window display provides the player information about the game status and helpful messages if they are stuck.

System Diagram



We decided to do all of our computation inside main.c, because there were not that many dependencies needed to external files. Utils.h was the only external dependency for setting up and toggling LEDs.

Software Description

There are three main files for our project. The main file runs the code and does the calculations for the map. Utils.c and utils.h both contain code from the class that set up and have LED functions available for use.

Within main.c, we perform most of our calculations for determining the position of a player, how much gold they have, whether they are blocked and other important game information. The structures and global variables in main are outlined below:

1. **typedef unsigned long realtime_t**
    a. This structure keeps track of the current time in the game. We use it to determine whether a player has been going along a current piece of the map for a long enough time to merit moving to a new piece.
2. **struct map_piece**
    a. Contains the primary structure for a map element, including gold, the duration of time that constitutes completing that piece and the exits from that piece. Exits is an array of 6 elements (ordered according to an enum direction_t which checks 6 different directions: up, down, left, right, pitch left, pitch right) which hold pointers to the next map piece that is accessible.
3. **Global variable ACCELEROMETER_STATE state**
    a. Contains the state of the accelerometer, an array of three elements for x,y and z positions.
4. **Global variable realtime_t current_time**
    a. Holds the current time of the game.
5. **Global variable realtime_t start_time**
    a. Holds the relative time a user has been inside a current piece.
6. **Global variable realtime_t base_duration**
    a. The base_duration a player must be in a given piece.
7. **Global variable map_piece_t * current_piece**
    a. Pointer to the current_piece that a player is in.
8. **Global variable map_piece_t * init**
    a. The starting piece of a map.
9. **Global variable map_piece_t * finish**
    a. The final piece of a map.
10. **Global variable int is_blocked**
    a. Flag to determine if the player is currently blocked on a path. 1 if blocked, 0 if not.
11. **Global variable int total_gold**
    a. The total gold a player currently has.

12. **Global variable int max_gold**
    a. The maximum gold a player can collect.
13. **Global variable LEDColor led_color**
    a. The current color the LED has been set to.

When an interrupt fires, most of the processing is done beforehand, so the interrupt handler simply updates the current time and resets the TFLG0 flag to 1. At the onset of a game, we initialize our hardware (accelerometer mainly) with hardware_init(), provided to us. We then proceed to initialize our LEDs and our accelerometer.

Once the hardware has been initialized, we need to initialize the state of the game. This means setting up the map given the specs provided and setting the global variable of current_piece to the initial piece of the maze. Finally, we call process_begin( ) to start the PIT timers, enable interrupts and enter into a continuous while loop to constantly update the game. Within this loop, we first need to get the current state of the accelerometer, and based on this, we can make calls to our helper functions to update the state of the game as necessary and give helpful print statements to the player.

Testing

Our testing was done incrementally with the helper functions that we wrote, given a specific testing map. Much of our code was not written until we had a thorough idea of the theory behind it. We then started testing with the board after completing all the specs for our project. This was done primarily by using randomly generated maps from our map generation function.

Results and Challenges

Unfortunately, we did not achieve the original project proposal that we had intended. We had severe hardware issues that caused us to pivot on our project.

For this new project, we had initial challenges in determining how we wanted to store our data, and what manner we wanted to organize the data. This went through several iterations before the design that we currently have. Another challenge that presented itself was the design of timers. Should we have timed interrupts or a continuous polling of the state of the accelerometer? In the end, because of the design of our map pieces as having duration, we decided it was best to fire an interrupt at given time intervals to better control and check how far a player has gone, or what they have completed etc.

Work Distribution

We worked as a group offline for the whole project, and coded together on the same laptop. We followed all peer programming techniques.

References and Software Reuse

We did not use any code from outside of this class. For the LED setup and functions,we used the code from Labs 3 - 5, namely the utils header and .c files.