fhwedel
UNIVERSITY OF APPLIED SCIENCES

# Maze Game

|  |  |
|---|---|
| **Name:** | **Chien-Hsun Chao (Tyler)** |
| **Student ID:** | **ITE104218** |
| **Professor:** | **Prof. Timm Bostelmann** |
| **Date:** | **14.02.2022** |

# Table of contents

# List of figures

# List of Tables

# 1. Introduction

## 1.1 Project description

The Maze game has a bit different from the original version. There are four levels with four different maps which are in a sequence of different difficulties. The idea of playing this game is that player should rotate the ball from the start area to the exit in a limited time. When the player passed the first level, the next level is coming up. And the game is ended until all four levels have been completed or the player failed.

In addition, Scores are calculated depending on the ending time. For instance, two points for one minute and one point for more than one minute. If the player does not pass the game because of hitting the wall, then there is no score for the game.

## 1.2 Project motivation

Maze game has always been popular in every generation. No matter what ages are the people, they all like to play the maze game. The maze game has been already developed in different versions and has been displayed in different ways. (e.g., 3D version, physical toys, and computer games). It can be seen from this; the maze game has a deep impression on people's minds. Therefore, I would like to implement this game on the microcontroller and discover how could this game become.

# 2. User manual

- Start
  Before starting the Maze game, the player should hold the MPU6050 in the horizontal mode to allow a ball to exist on the LCD screen. While the ball exists on the LCD screen, the player can start rotating the sensor to move the ball around.

- Wall Collision
  Once a ball collides with walls, the player should tilt the MPU6050 back to the ready position, which is the horizontal mode, and wait for the LCD to change the scenery.

- Goal
  Once the player rotated the ball to the goal area, the LCD screen will move on to the next level. The player should wait for two seconds for the next game to start. However, the player should also hold the MPU6050 to the ready position.

- All levels
  Once a player completes all levels of the game, the game will simply just show the final score and restart again from the first map. If a player would like to continuous playing the game, he or she should hold the MPU6050 in the horizontal mode for ready.

# 3 Project analysis

## 3.1 Modularization

The overview of the whole project is shown as the figure below.

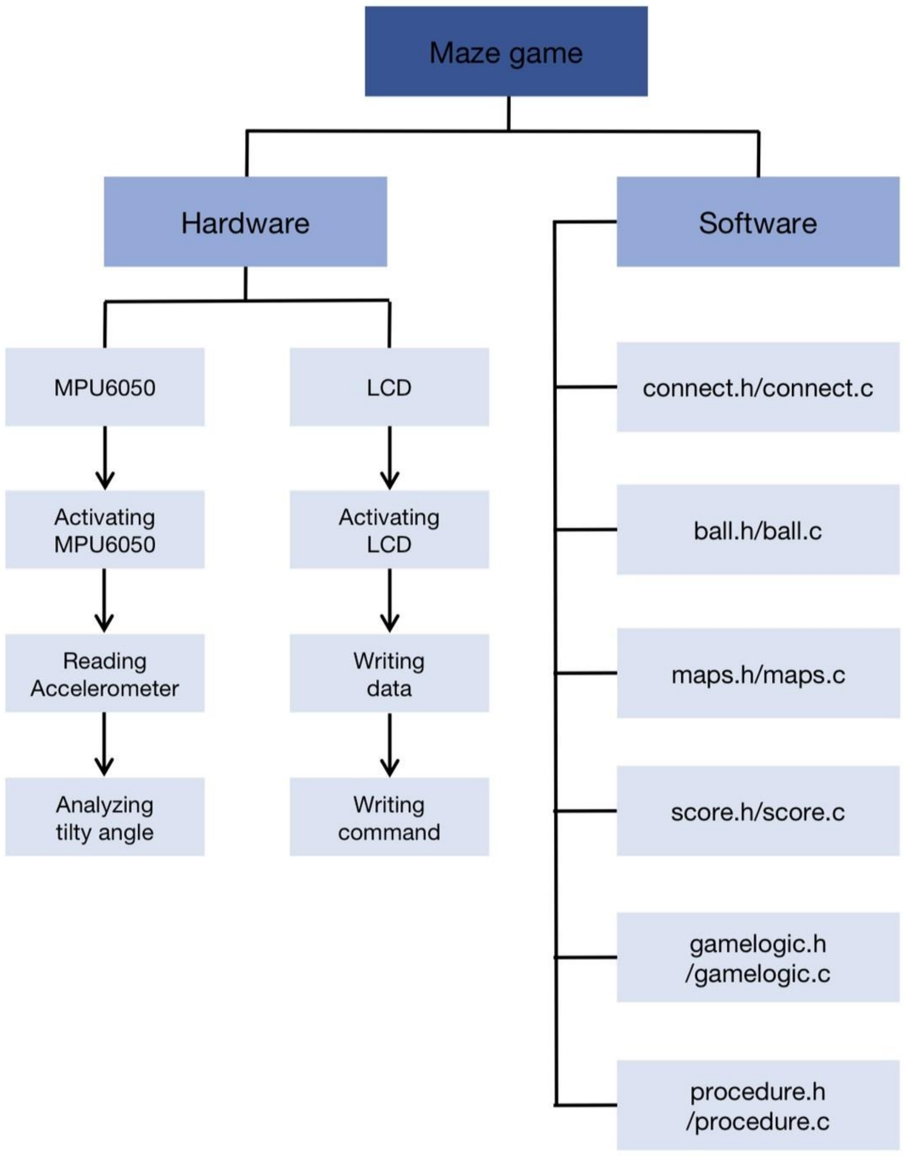


Figure 1: Overview of the maze game

## 3.1.1 Hardware

In this project, we will use three devices to implement our project, STM32F4 Nucleo-F401RE(Figure), LCD 128064b(Figure) and MPU6050 (Figure). The connection of each device is shown as the figure 2.

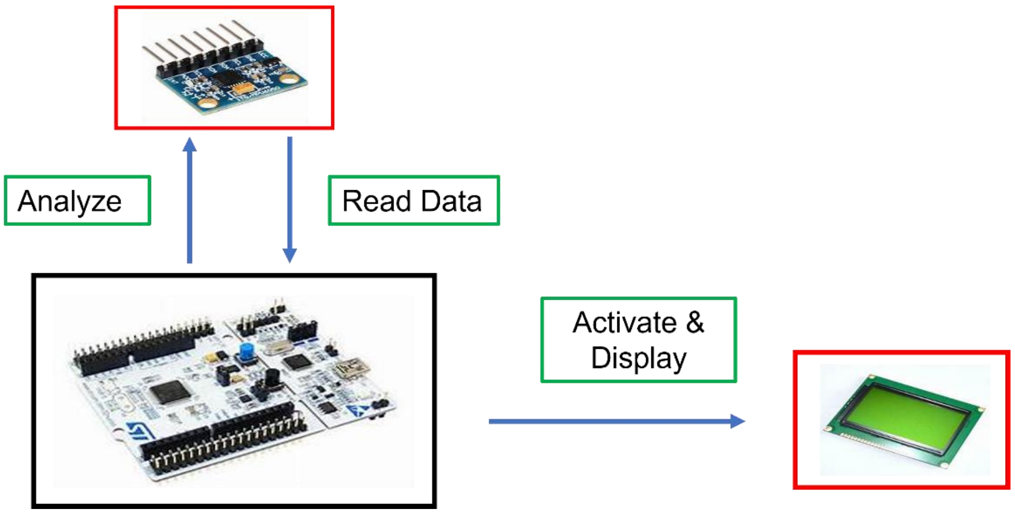


Figure 2: Hardware connection

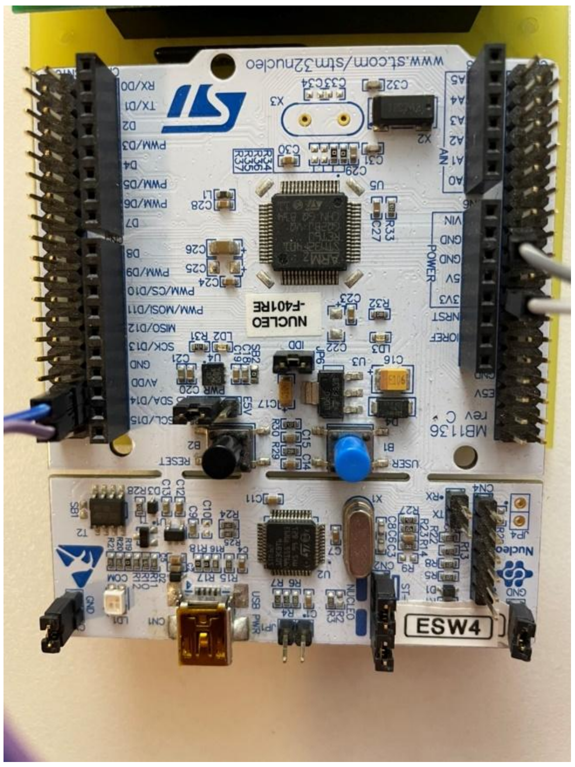


Figure 3: STM32F4 Nucleo-F401RE

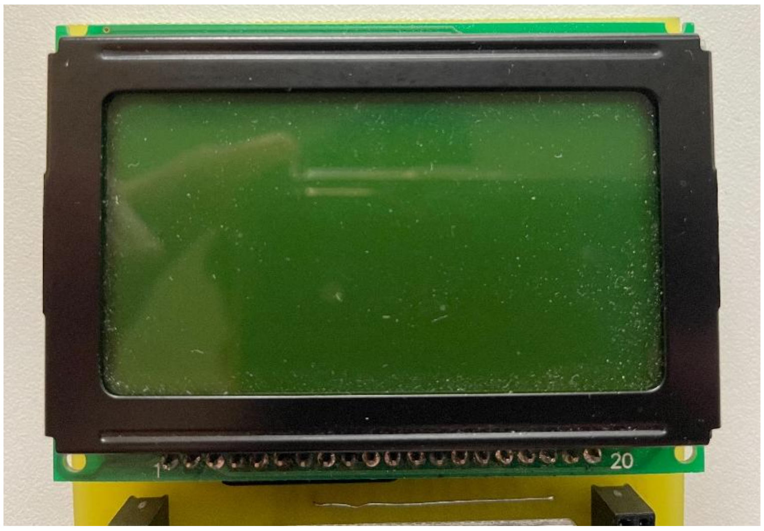


Figure 4: LCD 128064b

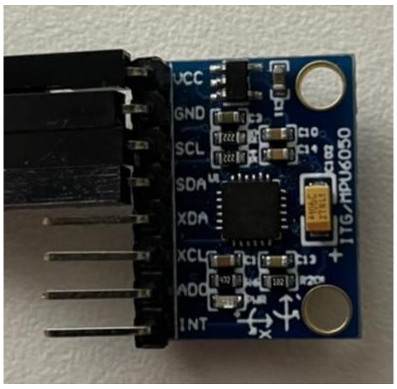


Figure 5: MPU6050

## 3.2 Abstract / algorithmic description

### 3.2.1 Analyzation of the accelerometer

In figure 6 below, we can use arcsine function to measure tilt angle with 2- axis of accelerometer. However, to accurately compute the angle of three axis of the accelerometer, we use the two formulae with three axis to calculate the roll (x) and pitch (y) angles.

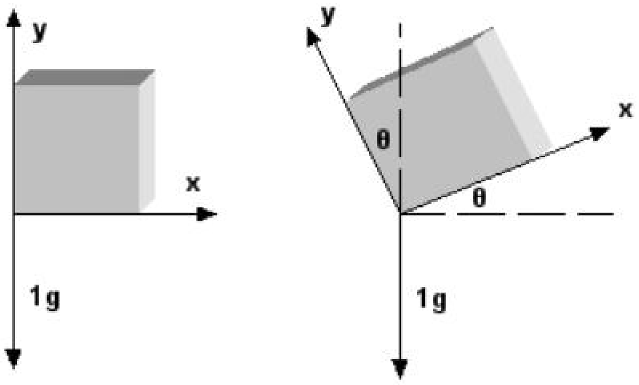
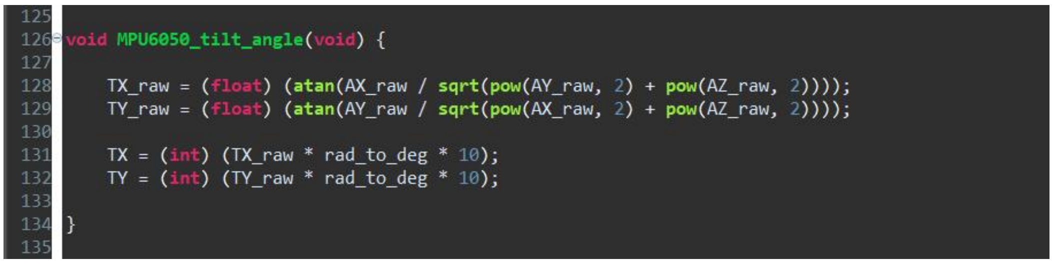


Figure 6: Titling the accelerometer

$$Ax = \arctan\left(\frac{X}{\sqrt{Y^2+Z^2}}\right)$$

$$Ay = \arctan\left(\frac{Y}{\sqrt{X^2+Z^2}}\right)$$

Figure 7: Formulae of roll (x) and pitch (y)



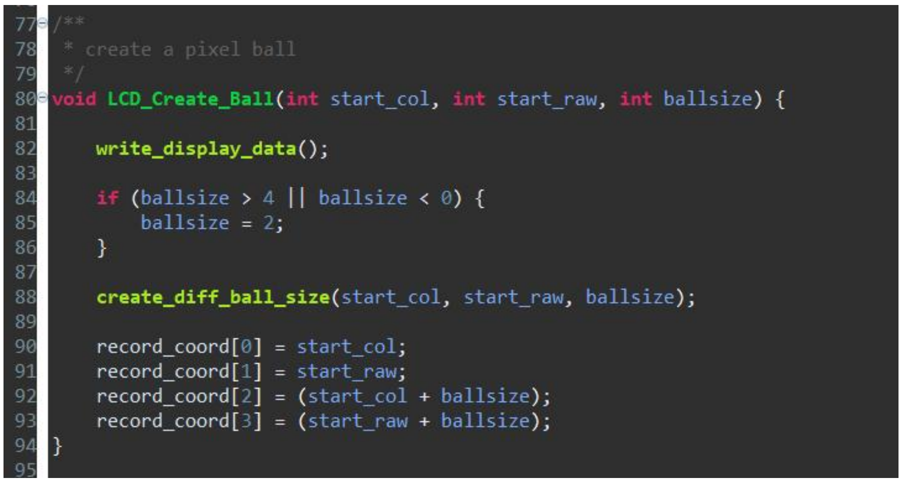
```
void MPU6050_tilt_angle(void) {

    TX_raw = (float) (atan(AX_raw / sqrt(pow(AY_raw, 2) + pow(AZ_raw, 2))));
    TY_raw = (float) (atan(AY_raw / sqrt(pow(AX_raw, 2) + pow(AZ_raw, 2))));

    TX = (int) (TX_raw * rad_to_deg * 10);
    TY = (int) (TY_raw * rad_to_deg * 10);

}
```

Figure 8: Code – Roll and Pitch

### 3.2.2 Ball

- Light up or off any pixels
  A function that lights up or lights off any required pixel with a specific column and row (e.g., LCD_draw_dot (uint8_t col, uint8_t row, uint8_t color)). In detail, a function of setting up pixel positions of the LCD screen is being called here to switch on or off a pixel and store the change in the RAM buffer which is a buffer array that will be used in the other following action or analysis.

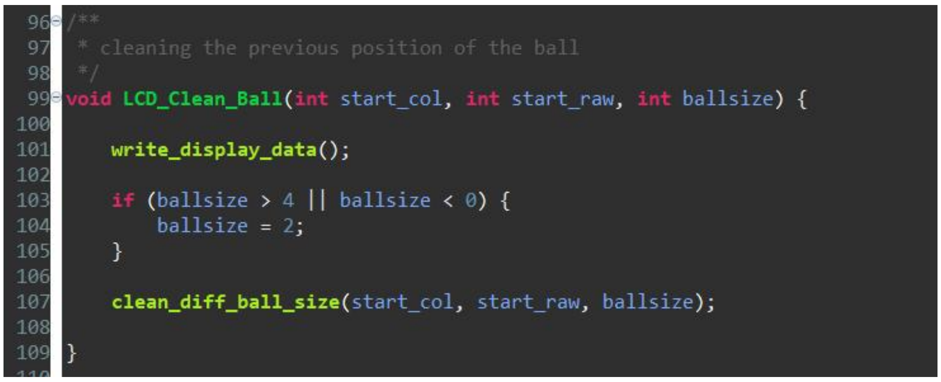
- Create a ball
  A function can create different sizes of a ball by giving the start position and a ball size (e.g., LCD_Create_ball (int start_col, int start_row, int ballsize)). There is a for-loop inside this function to increase a pixel along with the given ball size to form a proper ball from the given position. And because the created ball will be moved around on the map, its coordinate needs to be recorded in another new array (e.g., record_coord [4]) to be used to utilize in other following functions.



```c
/**
 * create a pixel ball
 */
void LCD_Create_Ball(int start_col, int start_raw, int ballsize) {

    write_display_data();

    if (ballsize > 4 || ballsize < 0) {
        ballsize = 2;
    }

    create_diff_ball_size(start_col, start_raw, ballsize);

    record_coord[0] = start_col;
    record_coord[1] = start_raw;
    record_coord[2] = (start_col + ballsize);
    record_coord[3] = (start_raw + ballsize);
}
```

Figure 9: Code – Create a ball

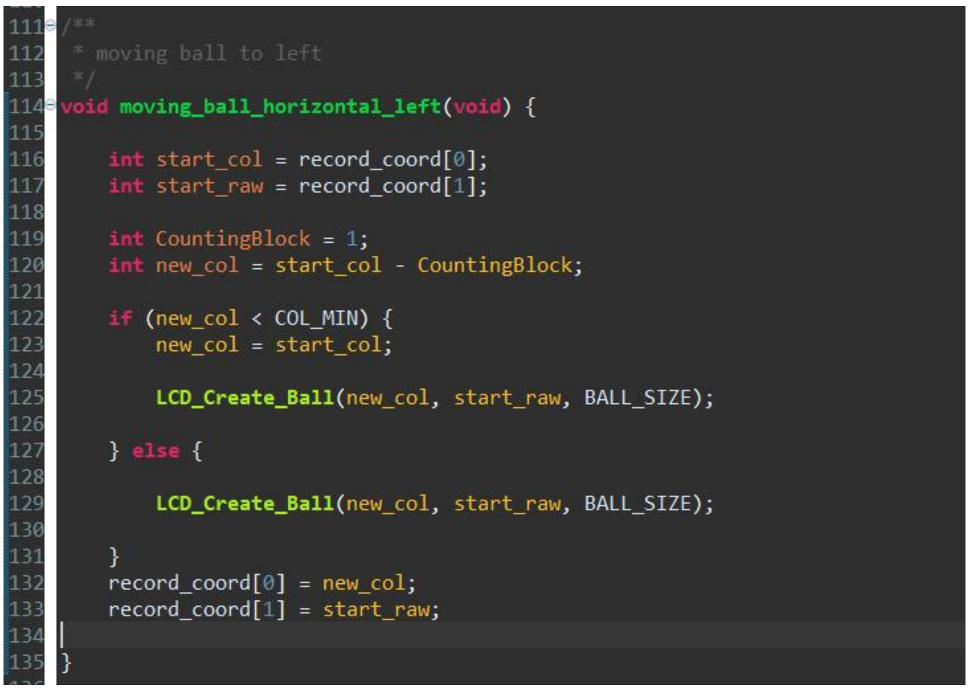
- Clear ball position
  This function mostly is utilized with the create ball function, because when a ball be created which means several pixels are being lighted up. While the ball is moving to another area, then the previous position of the pixel will not switch off automatically. Thus, a clear ball position function needs to be created and be used with the draw dot function with the argument of color 0.



```c
/**
 * cleaning the previous position of the ball
 */
void LCD_Clean_Ball(int start_col, int start_raw, int ballsize) {

    write_display_data();

    if (ballsize > 4 || ballsize < 0) {
        ballsize = 2;
    }

    clean_diff_ball_size(start_col, start_raw, ballsize);

}
```

Figure 10: Code – Clean a ball

- Moving in eight directions
  There are eight different functions to analyze a ball's direction. The concept of those functions utilizes the current coordinate of the ball to add a pixel respectively. And recording the coordinate of the adding pixel to move a ball. Moreover, we also check every boundary of the board in those functions to make sure the ball cannot be shown outside of the screen.



```c
/**
 * moving ball to left
 */
void moving_ball_horizontal_left(void) {

    int start_col = record_coord[0];
    int start_raw = record_coord[1];

    int CountingBlock = 1;
    int new_col = start_col - CountingBlock;

    if (new_col < COL_MIN) {
        new_col = start_col;

        LCD_Create_Ball(new_col, start_raw, BALL_SIZE);

    } else {

        LCD_Create_Ball(new_col, start_raw, BALL_SIZE);

    }
    record_coord[0] = new_col;
    record_coord[1] = start_raw;

}
```

Figure 11: Code – Roll and Pitch

# 4. Implementation

## 4.1 Specific description

- Ball hit walls:
  According to the maps we have designed, there are four different levels of the maps. To separate different maps, we use the switch case to distinguish four cases. In each case, "isTouchWall ()" is used to check if the ball's coordinates are as same as the coordinates of the walls. As the checking flag shows true, then" isTouchWall ()" will return the flag.

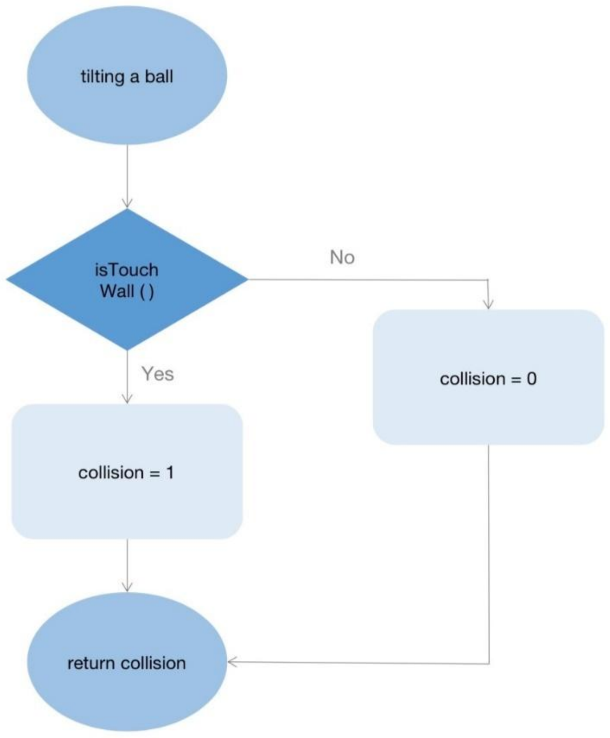


Figure 12: The flow diagram of touching walls

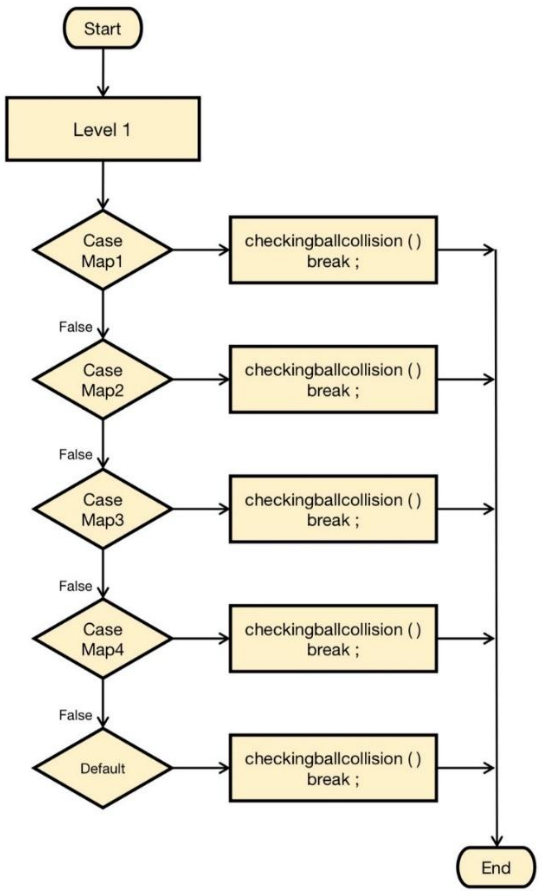

Start
Level 1
Case Map1
Case Map2
Case Map3
Case Map4
Default
False
False
False
False
checkingballcollision ( ) break ;
checkingballcollision ( ) break ;
checkingballcollision ( ) break ;
checkingballcollision ( ) break ;
checkingballcollision ( ) break ;
End

Figure 13: The flow diagram of checking different maps

- Ball reach goal:
  When a ball is rotated in the exit, the game is completed. The way of recognizing if the ball reaches the goal or not is that comparing the ball's coordinate and the wall's coordinate. So, "isBallreachGoal ()" is created for this purpose.

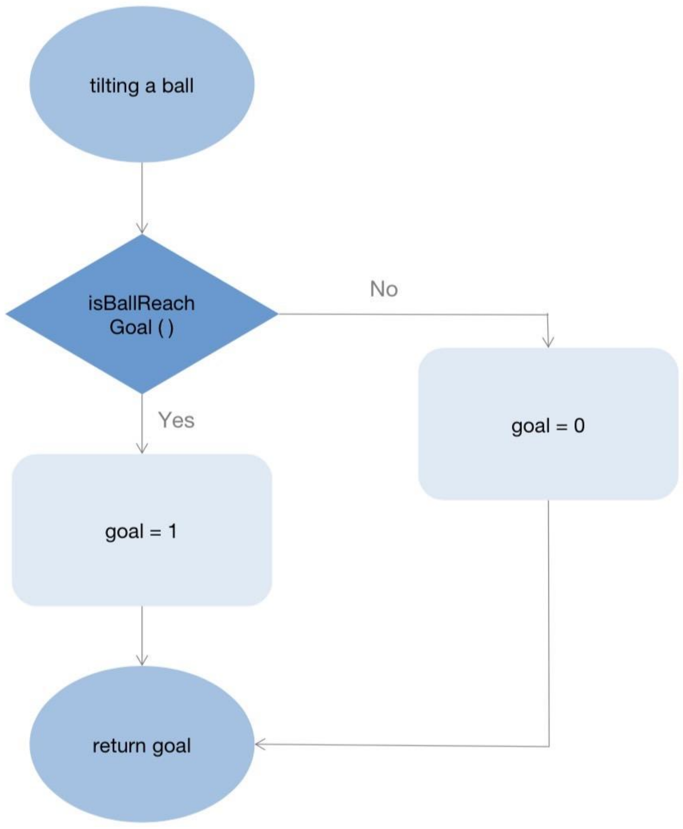
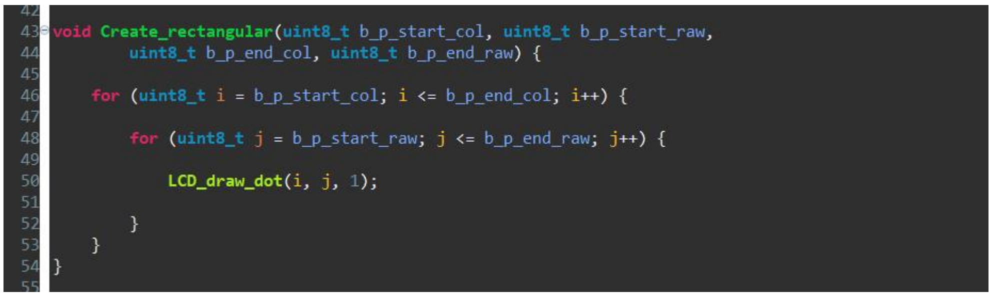


Figure 14: The flow Diagram of reaching goal

- Creating a map:
  According to the pattern that I design, I will create a function for creating a rectangular. Every map can depict its patterns by calling this function. And the concept of this function is using the two boundary points to calculate the whole area of a block. In addition, there are more than one rectangular in one map. So, the for-loop is used to run with a map array storing every coordinate of walls.



```c
void Create_rectangular(uint8_t b_p_start_col, uint8_t b_p_start_raw,
        uint8_t b_p_end_col, uint8_t b_p_end_raw) {

    for (uint8_t i = b_p_start_col; i <= b_p_end_col; i++) {

        for (uint8_t j = b_p_start_raw; j <= b_p_end_raw; j++) {

            LCD_draw_dot(i, j, 1);

        }
    }
}
```

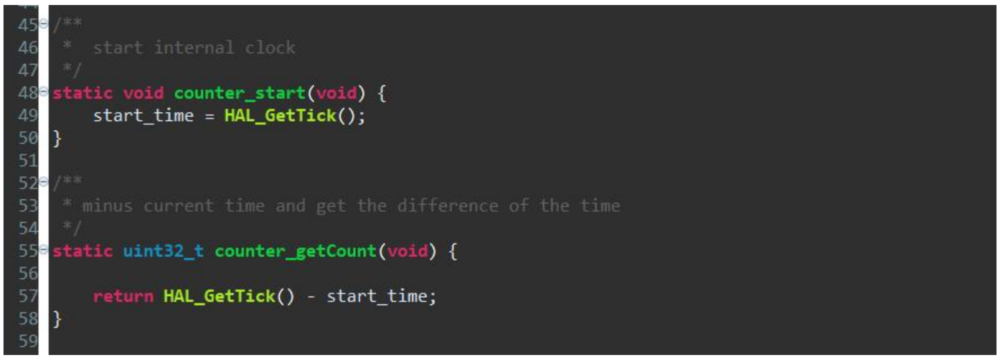
Figure 15: Code - Create rectangular

- Changing to next scene

  When the ball arrived at the specific area which is the exit, the whole screen will turn black for one second and turn to white the next second then change to the next level. Thus, there is a function" ChangScenary ()" for turning the whole screen to black and to white.

- Create a counter

  To acquire the final score depending on when the player completes a game, we need a global counter to record the time in each map. Then, we decide to get the time from HAL_GetTick () function However, it only provides the uptime of the whole system. This means that when the HAL_GetTick () is called, the time starts counting. But we need a proper time value to calculate the final score. Thus, we use another function called get_counter () calling the HAL_GetTick () again, and we will get the difference between the previous HAL_GetTick () and the one called later, and this difference is the time value.



```c
/**
 *  start internal clock
 */
static void counter_start(void) {
    start_time = HAL_GetTick();
}

/**
 * minus current time and get the difference of the time
 */
static uint32_t counter_getCount(void) {

    return HAL_GetTick() - start_time;
}
```

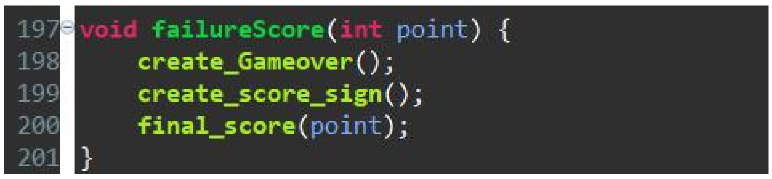
Figure 16: Code – Create a counter and get time difference

- A score function

  This function "cal_score ()" is used to calculating the score. The way of calculating will be like 2 points for one minute, 1 point for more than one minute. The last return value will be the final score.

- Failure sign

  An array is created to store the coordinate of few words and the scores. It will be implemented in an independent function “FailureSign ()”.

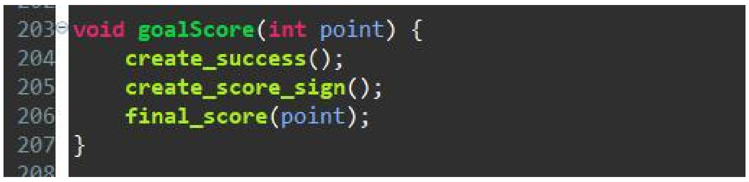


```
void failureScore(int point) {
    create_Gameover();
    create_score_sign();
    final_score(point);
}
```

Figure 17: Code – Create failure sign

- Success sign

  An array is created to store the coordinate of few words, the scores. It will be implemented in an independent function “SuccessSign ()”.



```
void goalScore(int point) {
    create_success();
    create_score_sign();
    final_score(point);
}
```

Figure 18: Code – Create success sign

- Manage every progress

  A function for managing every process calling “gameRunning ()”. This function includes the two conditions “isTouchWall ()” and “isBallReachGoal ()”. Those two conditions will be checked when the ball starts tilting. When the ball collides with a wall, then the scenery is changing, and the failure sign is called. In another condition that a ball reaches the goal, the function of changing scenery is also called, and the success sign will exist here.

## 4.2 Software Structure

Here are the files that mostly contain the whole project and each file has its own purpose for completing a whole project.

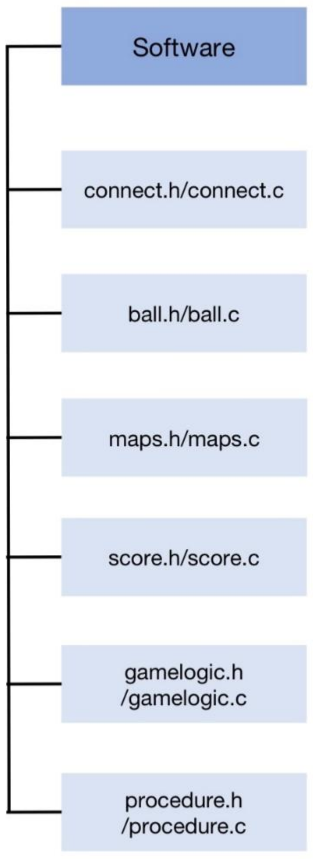


Figure 19: All files

- ball.h / ball.c
  This file includes several functions to create a ball on the LCD screen. For example, a function LCD_draw_dot () to light up or light off a pixel. And a function to create a ball with the required size and a function to clear every position of the ball. The last, there are eight functions to guide the direction of the ball when it is being moved.

- connect.h / connect.c
  The file connects the ball file and other files. There is the main function called tilting () which analyzes the action of the ball. It will be called in the app_loop () function. And there is another function to initialize the ball position when it is standstill, and the accelerometer is not moving.

- delay.h / delay.c
  This is file is created for a delay function. Due to the reason that the HAL_DELAY only provides the millisecond delay, the activation of LCD needs a microsecond delay to initialize the device. Thus, delay_us () is created.

- gameLogic.h / gameLogic.c
  This file contains every function that is used to sort the whole game logic. In fact, as the game start and the ball start to be rotated, it will touch walls or reaches the goal area. Since it touches the walls or arrives at the goal, the score should be calculated depending on the playing time. In addition, as the scenery is changed from the map to the map or map to displaying the score, an interface of changing the scenery provides a better understanding of the game situation. Thus, those conditions of those actions are required and need to be completely set up before they are being called.

- lcd.h / lcd.c
  This file contains several functions of activating LCD devices. According to the LCD user manual and datasheet, there are a few pin settings needed to be initialized, and a few specific registers needed to be known and be executed. For instance, if the data is expected to be written into the device and be shown on the screen, then a few pins of the LCD device need to be set up as a specific value.

- maps.h / maps.c
  Due to the maps being designed by the designer, the walls in the maps are fixed in the specific coordinates also their exits are fixed. Therefore, we still need a few functions to show these walls and exits on the LCD screen. In general, the purpose of this file is that create maps that are expected and declare several arrays to store the coordinate of those maps and exits.

- mpu6050.h /mpu6050.c
  The project is completed using mpu6050 to detect the accelerometer rotating difference to control how is a ball tilting around. So, this file plays an important role that analyzing the accelerometer and calculating its titling angle by three axes. It shows these measured values on the laptop to efficiently realize how the direction of the ball should be set up.

- procedure.h / procedure.c
  This file is responsible to manage the game start and executing the whole game by checking goal condition and wall condition. Literally, it managed the whole process which means deciding how should the game be set up.

- score.h / score.c
  As the game begins to the end, whatever how is the result go, there must be a score to present the player's effort. The way of doing this is to score, and this file mainly manages the final score. It contains many arrays of signs and functions to create different scores of signs.

## 4.3 Schematic

Here is some Schematic for four different levels of the map and the display of the win and lose game.

### 4.3.1 Four levels of the maps

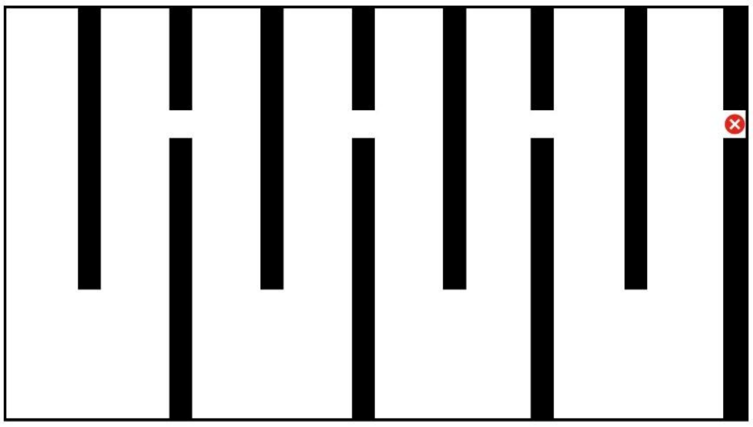


Figure 20: Map 1

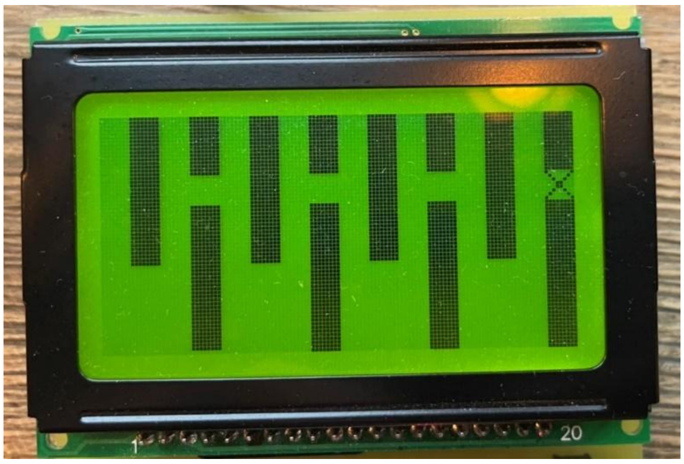


Figure 21: LCD_map 1

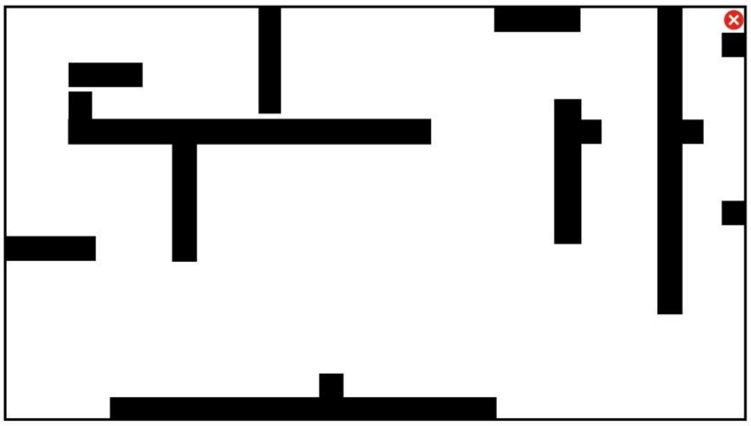


Figure 22: Map 2

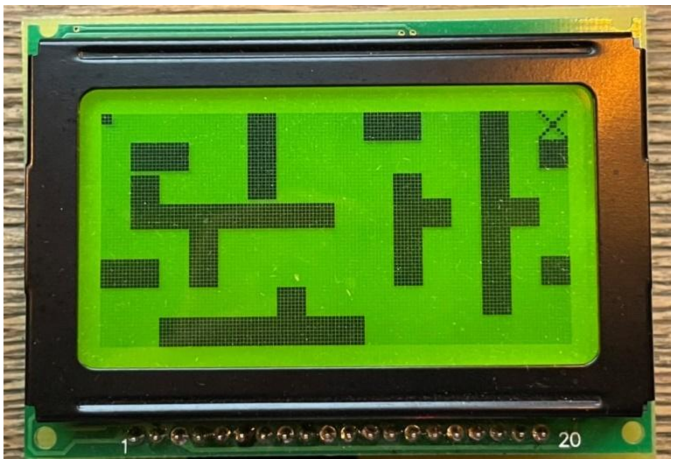


Figure 23: LCD_map 2

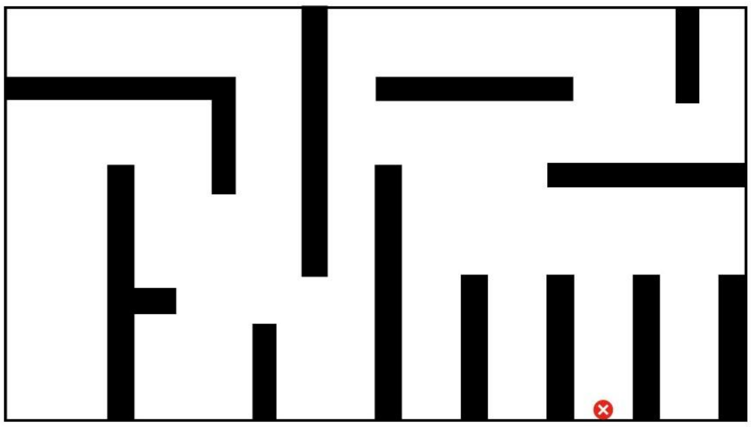


Figure 24: Map 3

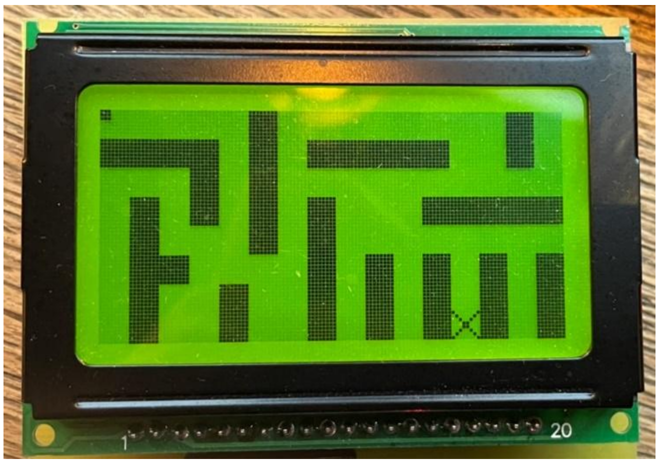


Figure 25: LCD_map 3

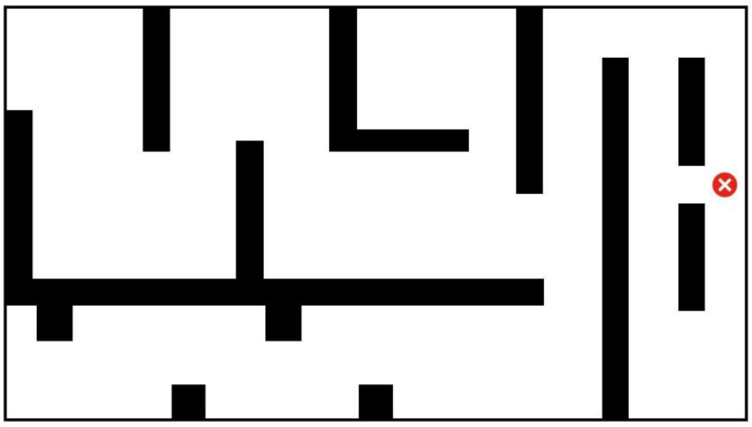


Figure 26: Map 4

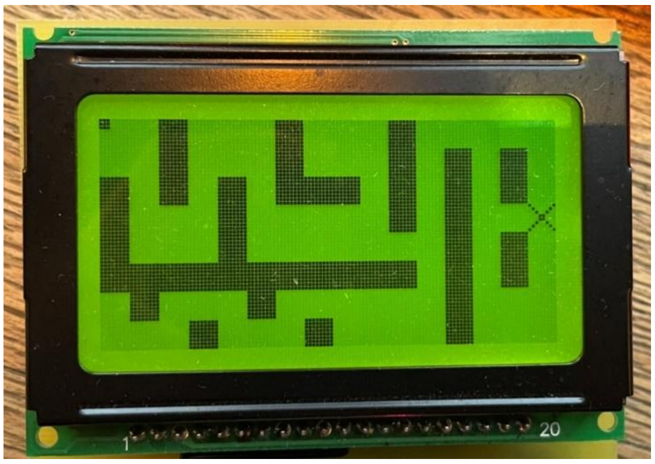


Figure 27: LCD_map 4

## 4.3.2 Score sign

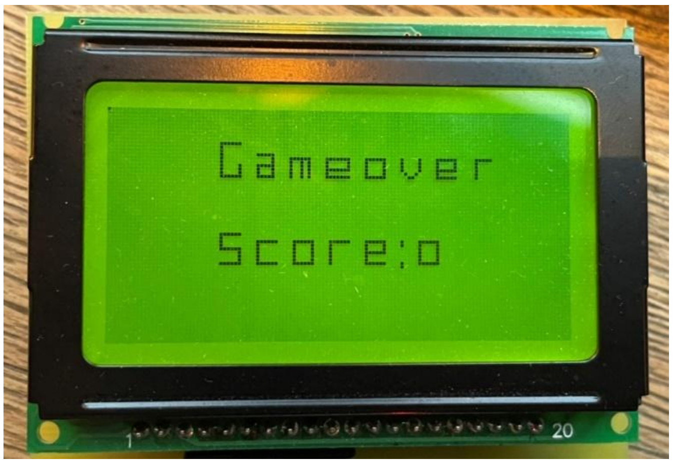


Figure 28: Failure Sign

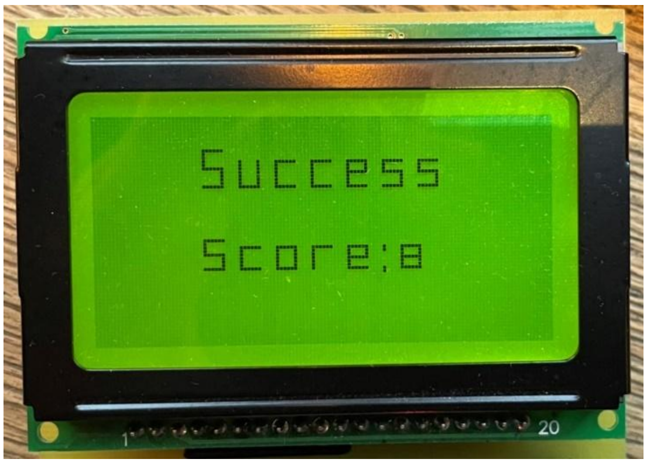


Figure 29: Success Sign

# 5. Test

## 5.1 The testing of walls collision

The purpose of this testing is to check when the ball collides with each wall of four maps, and the whole game will start at the beginning.

- Map 1

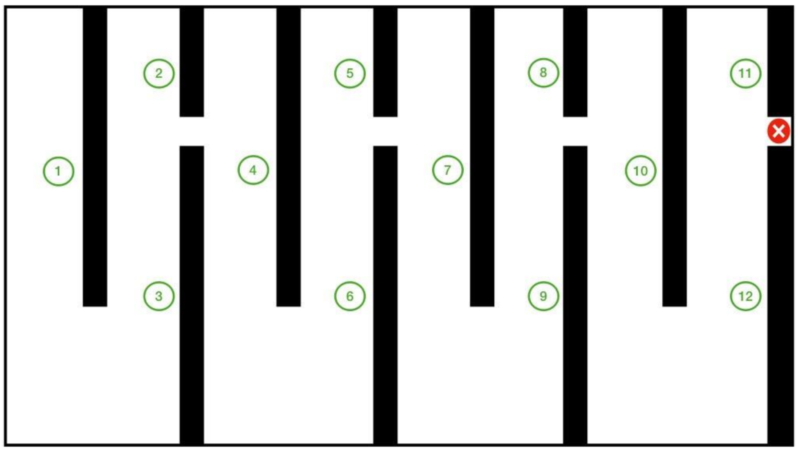


Figure 30: The map_1 with walls number

| | |
|---|---|
| ✓ Wall 1 | ✓ Wall 2 |
| ✓ Wall 3 | ✓ Wall 4 |
| ✓ Wall 5 | ✓ Wall 6 |
| ✓ Wall 7 | ✓ Wall 8 |
| ✓ Wall 9 | ✓ Wall 10 |
| ✓ Wall 11 | ✓ Wall 12 |

Table 1: The walls of map 1

- Map 2

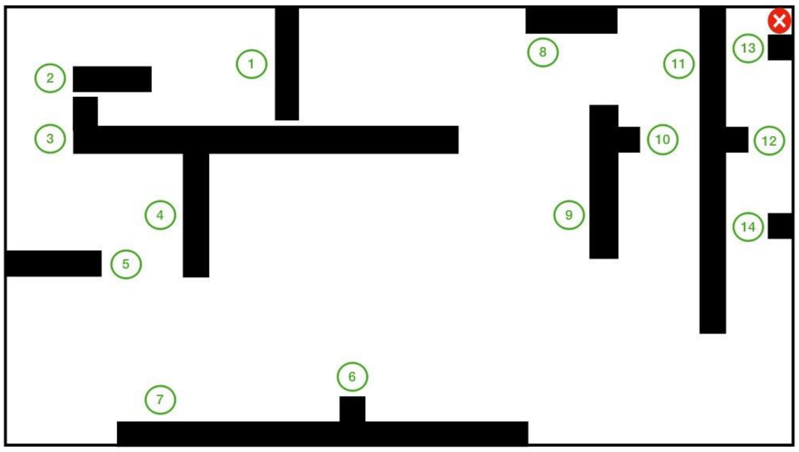


Figure 31 : The map_2 with walls number

| | |
|---|---|
| ✓ Wall 1 | ✓ Wall 2 |
| ✓ Wall 3 | ✓ Wall 4 |
| ✓ Wall 5 | ✓ Wall 6 |
| ✓ Wall 7 | ✓ Wall 8 |
| ✓ Wall 9 | ✓ Wall 10 |
| ✓ Wall 11 | ✓ Wall 12 |
| ✓ Wall 13 | ✓ Wall 14 |

Table 2: The walls of map 2

- Map 3

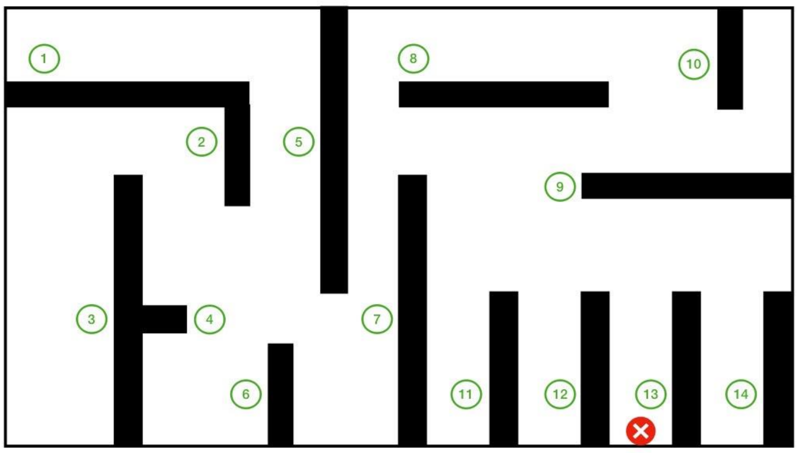


Figure 32: The map_3 with walls number

| | |
|---|---|
| ✓ Wall 1 | ✓ Wall 2 |
| ✓ Wall 3 | ✓ Wall 4 |
| ✓ Wall 5 | ✓ Wall 6 |
| ✓ Wall 7 | ✓ Wall 8 |
| ✓ Wall 9 | ✓ Wall 10 |
| ✓ Wall 11 | ✓ Wall 12 |
| ✓ Wall 13 | ✓ Wall 14 |

Table 3: The walls of map 3

- Map 4

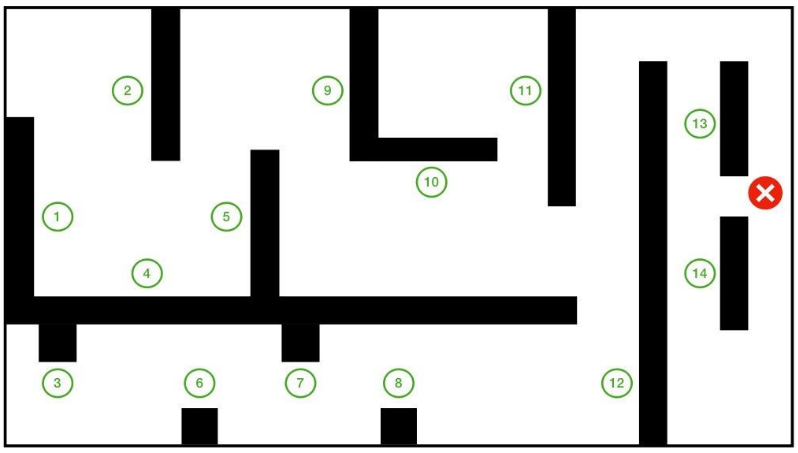


Figure 33: The map_4 with walls number

| | |
|---|---|
| ✓ Wall 1 | ✓ Wall 2 |
| ✓ Wall 3 | ✓ Wall 4 |
| ✓ Wall 5 | ✓ Wall 6 |
| ✓ Wall 7 | ✓ Wall 8 |
| ✓ Wall 9 | ✓ Wall 10 |
| ✓ Wall 11 | ✓ Wall 12 |
| ✓ Wall 13 | ✓ Wall 14 |

Table 4: The walls of map 4

## 5.2 The Score Testing

About the score testing, because there are four levels and a time limit in each game. According to the completed time, the player will get different points.

In the table below, we generally show different levels, and the sum of the points player will get.

| Level 1 | | Level 2 | | Level 3 | | Level 4 | | Sum | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| X | | | | | | | | 0 | | | | |
| o | 1 | x | | | | | | 1 | 2 | | | |
| | 2 | | | | | | | | | | | |
| o | 1 | o | 1 | x | | | | 2 | 3 | 4 | | |
| | 2 | | 2 | | | | | | | | | |
| o | 1 | o | 1 | o | 1 | x | | 3 | 4 | 5 | 6 | |
| | 2 | | 2 | | 2 | | | | | | | |
| o | 1 | o | 1 | o | 1 | o | 1 | 4 | 5 | 6 | 7 | 8 |
| | 2 | | 2 | | 2 | | 2 | | | | | |

Table 5: All possible score points

According to the Table 5, the total score of the game is 8 points. By different combination of different levels, players can get 1 or 2 points from every level if the game is success. And 0 points when the ball hit the walls.

- Score 0

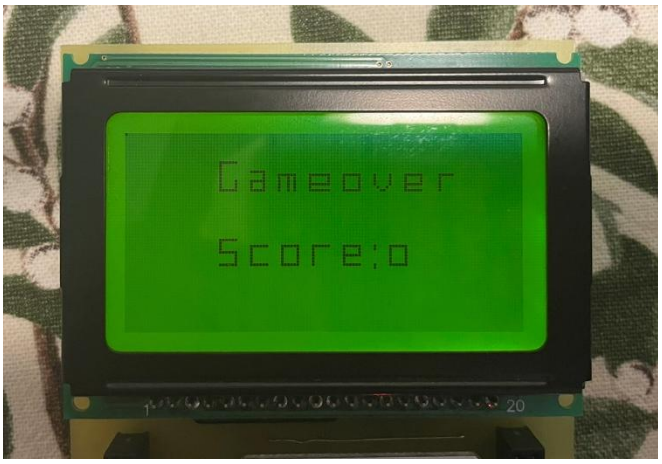


Figure 34: Score 0

- Score 1

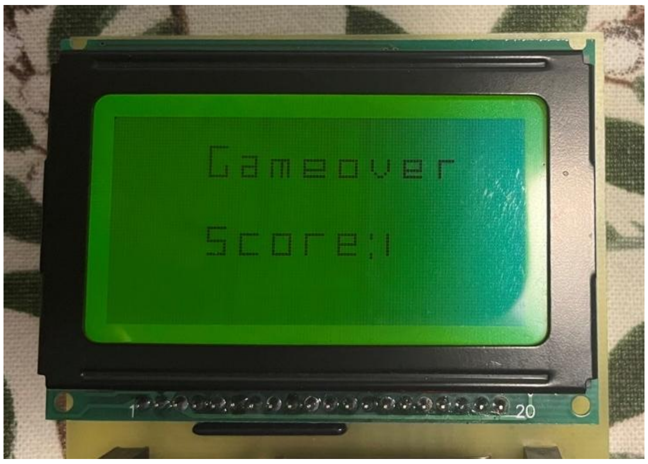


Figure 35: Score 1

- Score 2

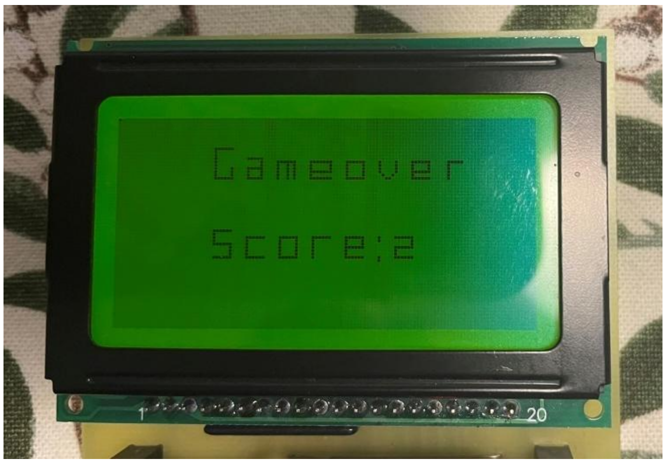


Figure 36: Score 2

- Score 3

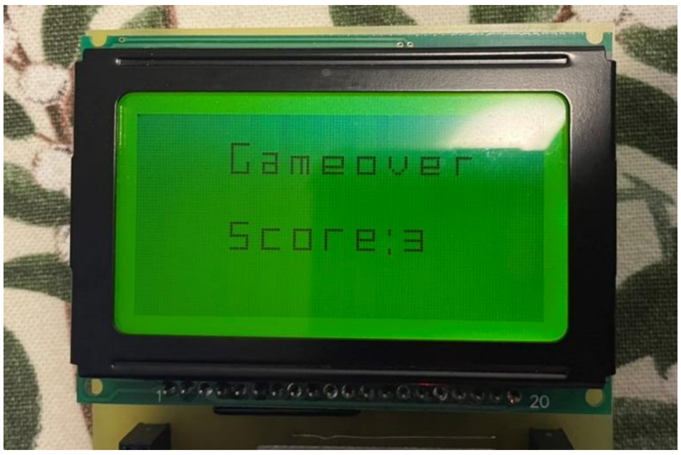


Figure 37: Score 3

- Score 4

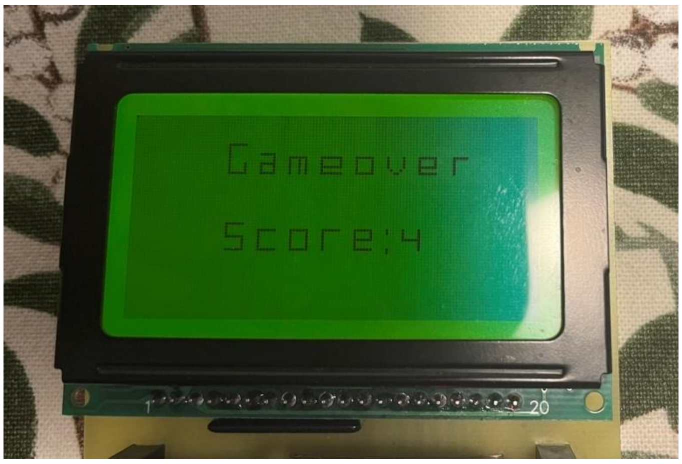


Figure 38: Score 4

- Score 5

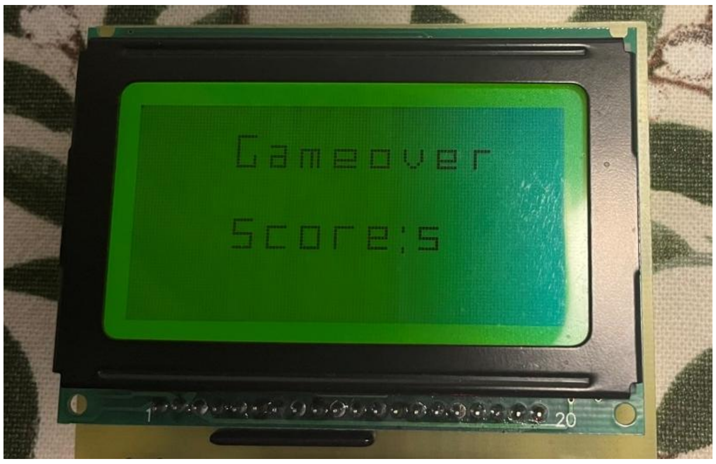


Figure 39: Score 5

- Score 6

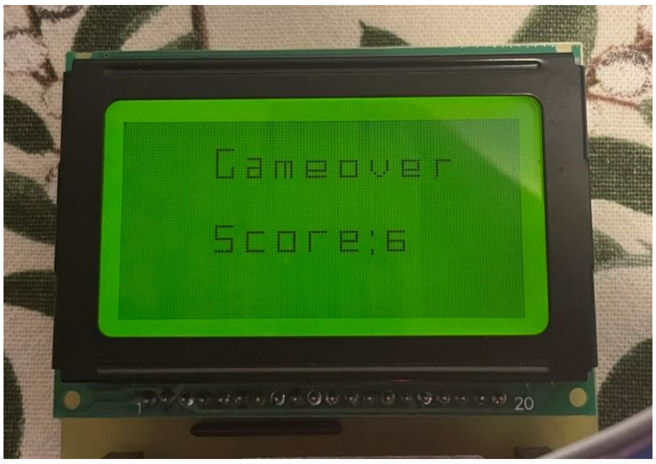


Figure 40: Score 6

- Score 7

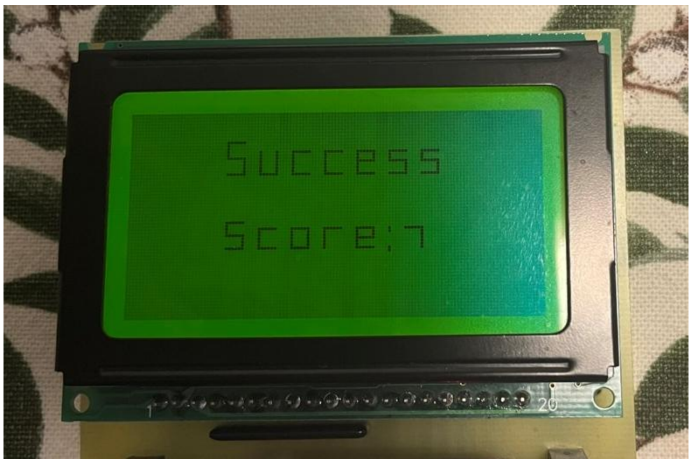


Figure 41: Score 7

- Score 8

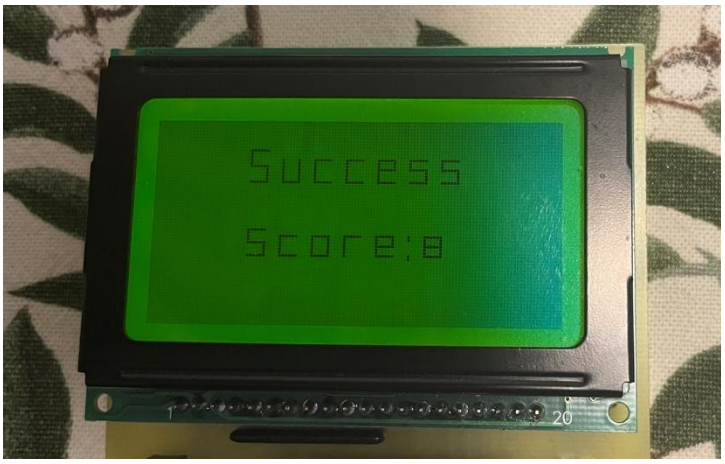
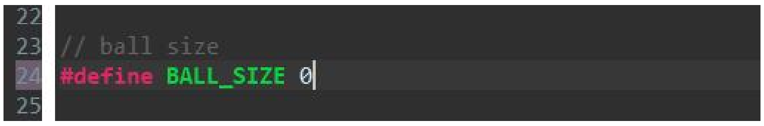


Figure 42: Score 8

## 5.3 The Ball size

In this game, the ball size is allowed to increase if the player would like to increase the difficulties of the game. However, there is still a limit of the ball size. It must not less than 0 because there must be a ball exist one the screen. And the ball must not more than size 4 which means it cannot be larger than 4 pixels. Here we will put some test related to increasing ball size for 0 to 4.

- Ball size 0



```
// ball size
#define BALL_SIZE 0
```

Figure 43: Code - the Ball size 0

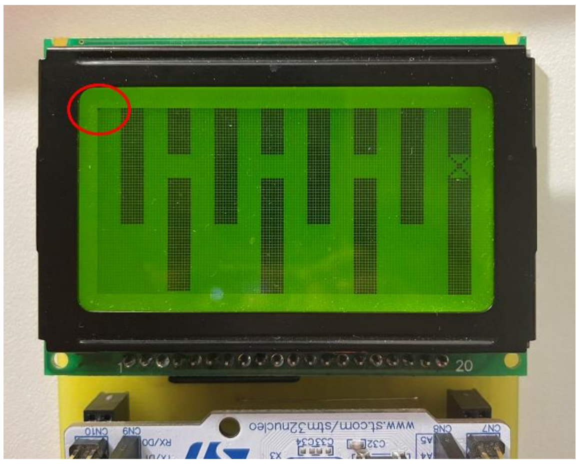
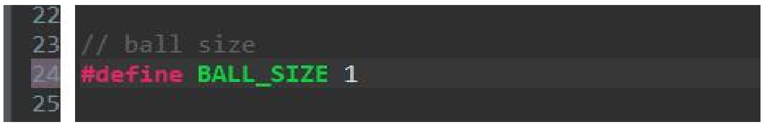


Figure 44: The Ball size 0

- Ball size 1



```
// ball size
#define BALL_SIZE 1
```

Figure 45: Code - the Ball size 1

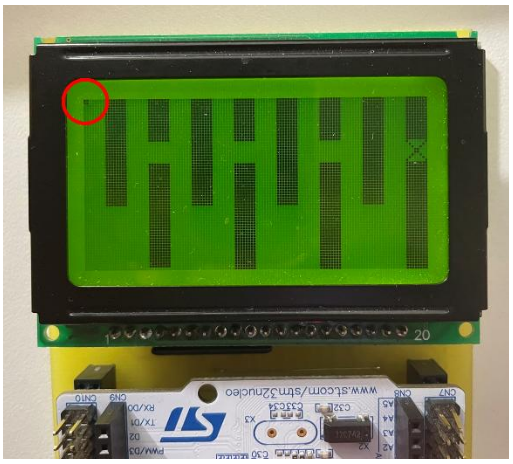
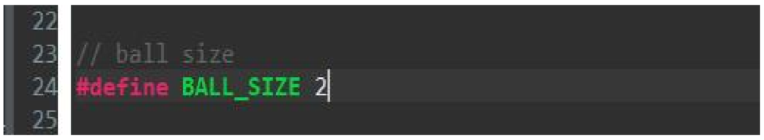


Figure 46: The Ball size 1

- Ball size 2



```
// ball size
#define BALL_SIZE 2
```

Figure 47: Code - the Ball size 2

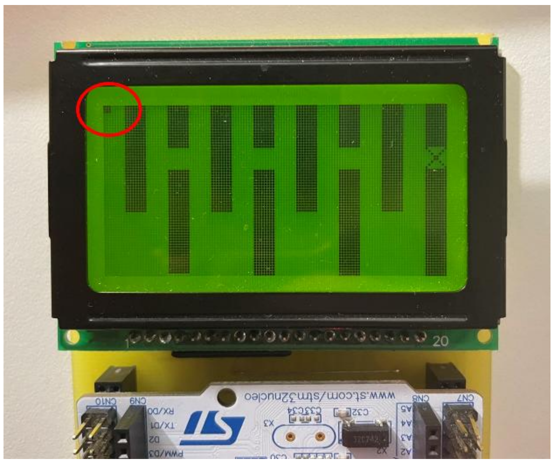
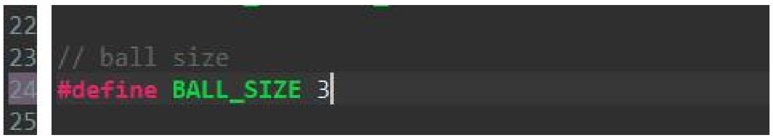


Figure 48:The Ball size 2

- Ball size 3



```
// ball size
#define BALL_SIZE 3
```

Figure 49: Code – the Ball size 3

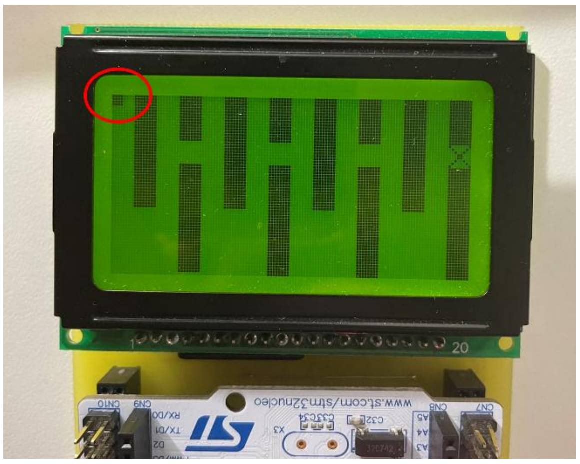
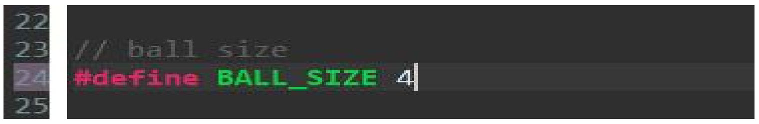


Figure 50:The Ball size 3

- Ball size 4



```
// ball size
#define BALL_SIZE 4
```

Figure 51: Code - the Ball size 4

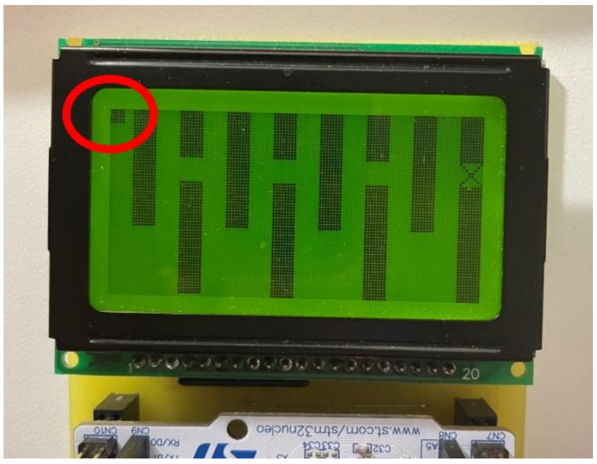


Figure 52: The Ball size 4

## 6. Conclusion

Through this project implementation, I believe that there are more methods to implement the maze game. And there is more idea of designing different maze games. This project could be extended to more aspects like adding different conditions or changing the game style. Even the game is not fully implemented as the original setting, it still presents a different and interesting way.

## 7. Reference

- https://www.hobbytronics.co.uk/accelerometer-info