

# **IERG3810**

## **Microcontrollers and Embedded System Lab**

### **Project Report**

#### **Quick Calculation Game with Skills**

**Group: B12**



**Members: KEI Yat-long**

**Li Hong Man**

**Submission Date: 23/4/2021**

#### **Disclaimer**

I declare that the assignment here submitted is original except for source material explicitly acknowledged, and that the same or related material has not been previously submitted for another course. I also acknowledge that I am aware of University policy and regulations on honesty in academic work, and of the disciplinary guidelines and procedures applicable to breaches of such policy and regulations, as contained in the website <http://www.cuhk.edu.hk/policy/academichonesty/>

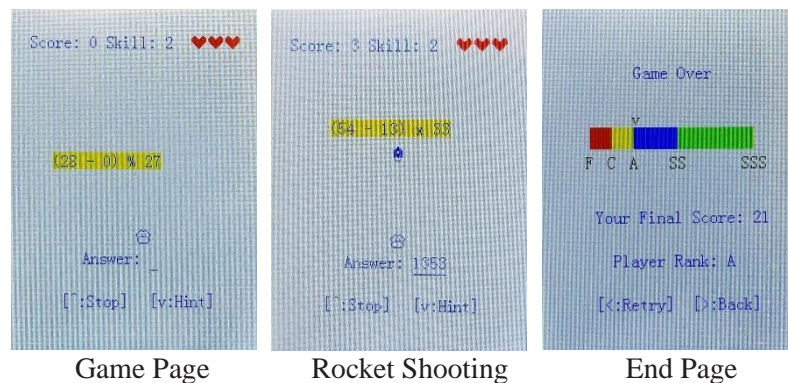
	KEI Yat-long	23/4/2021
Signature	Name	Date
	LI Hong Man	23/4/2021
Signature	Name	Date

## I. OBJECTIVES

The objective of this project is to design an electronic game which implements all the knowledge we learnt in Lab-1 to Lab-5 and demonstrates our practical programming skills.

## II. GAME INSTRUCTION

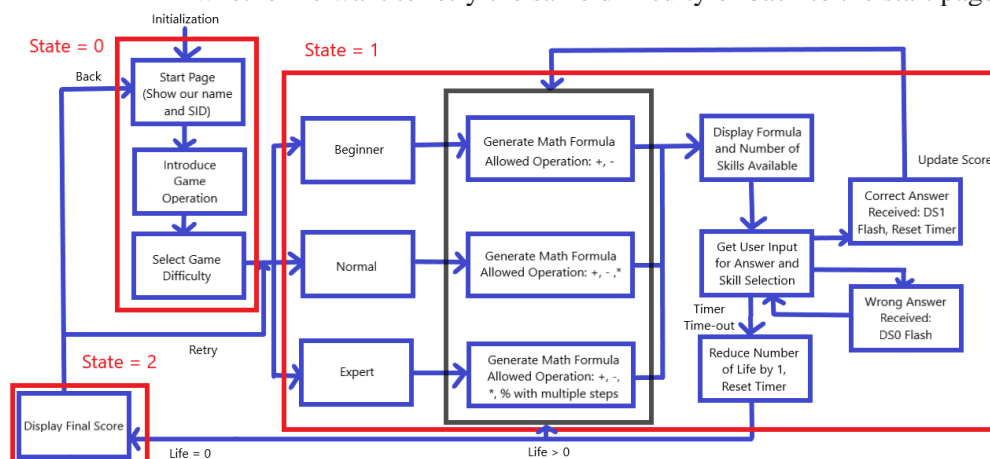
Our game is called “Quick Calculation Game with Skills”, in which users need to calculate math equations within a limited time. In this game, text blocks with equations will keep generating at the upper part of screen and falling down as time evolves, players need to destroy the question blocks before they reached the bottom of main screen. Initially, player has 3 lives, the number of lives will be decremented by one for each timeout event. The complexity of equations depends on the difficulty level chosen by the player. To destroy a block, players need to make the icon at the bottom of text block, input correct answer of equation with numpad keyboard, and then click “ENTER” to shoot a rocket. Players can choose to use skills which can stop the timer for 2 seconds or give the 1st digit answer when they are stuck in that question. When player use up their lives, program terminates and display the final score and ranking on the screen. For example, if the score is lower than 3, then is ‘F’ grade. Player can press the “\*” key at any time to reset the game.



## III. PROGRAM FLOW CHART

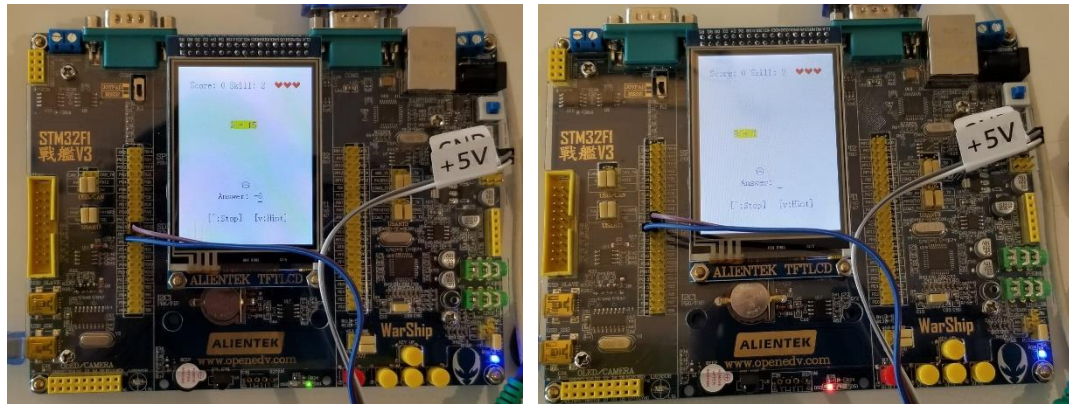
Our program flow can be modeled as a finite state machine with 3 different states. State transitions are triggered by user input and the game timer.

- State = 0: **Preparation stage.** User can check the rule of game and select difficulty.
- State = 1: **Game stage.** Main game logic is executed. User can move user icon, input answers, and make use of skill, screen is updated to provide animation that makes the game more interactive.
- State = 2: **Termination stage.** Score and rank of user are displayed, user can select whether he want to retry the same difficulty or back to the start page.



#### IV. HARDWARE FEATURES

For **LED/BUZZER/BUTTON**, we implemented in the main game. When the player answers the question correctly, LED1 (Green) flashes. Alternatively, if the player answers wrongly, or the rocket does not hit the text block, LED0 (Red) flashes.

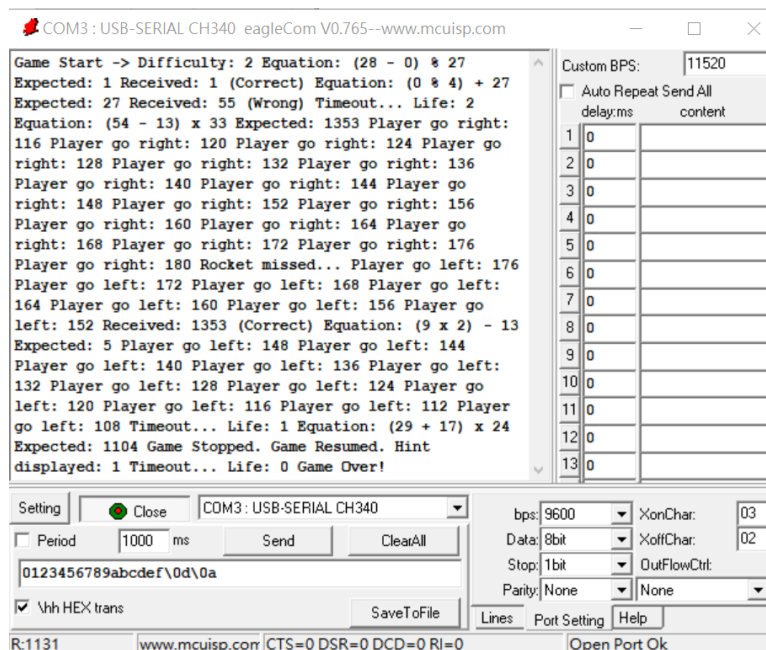


Correct answer is received

Wrong answer is received or rocket missed

Player also needs to press buttons to control the character movement and use the skills. When game is over, the buzzer will ring.

For **USART-2**, we implemented as an instruction for player to know what is happening in the game now. For example, when the game start, USART shows "Game Start ->" on the screen. This is also useful for debug purpose.



Sample Output from USART-2

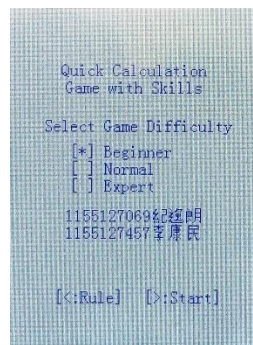
For **LCD** display, we implemented as the place for the game to show. We show everything like the rule, animation, ranking, etc. on the screen.

For **Interrupt**, we implemented as button click and keyboard pressing, as well as the stop skill in the game. The player mainly press keyboard to answer questions or reset the game.

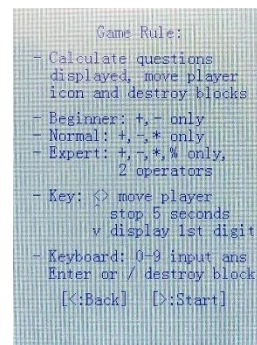
For **Timer & SysTick**, we implemented as a counter to check the timeout of the game. This helps us to concisely control the time limit of each question.

## V. DESCRIPTION OF FUNCTIONALITIES AND FEATURES

In start page, user can press the left key to switch between difficulty selection page and rule page. Also, User can press the right key to start a new game.



Start Page

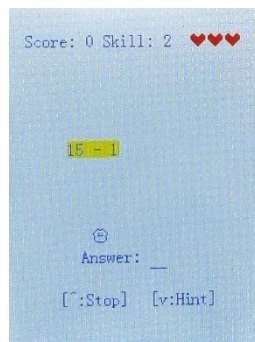


Rule Page

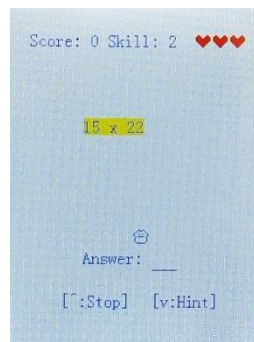
The details of game page and end page has been explained in previous sessions.

This game supports selection of **3 difficulty levels**:

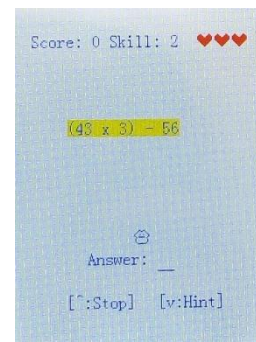
- Beginner: only operation “+” and “-” is included.
- Normal: in addition to beginner, “x” is included.
- Expert: in addition to normal, “%” (modulus) is included, and there are 2 operations in each equation.



Beginner



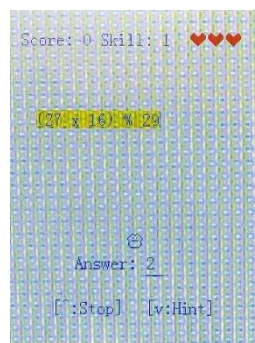
Normal



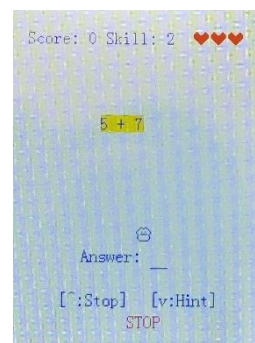
Expert

There are also **2 skills** available for players:

- Hint: display the first digit of answer, cannot be used if it only has 1 digit.
- Stop: stop the whole game by 2 seconds, a “STOP” message is shown.



Hint Skill



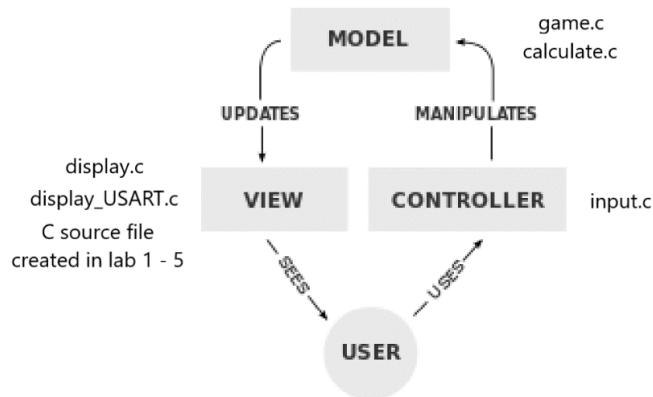
Stop Skill



## VI. SOFTWARE DESIGN

### Design Pattern

In high level, we followed the **Model-View-Controller pattern** that enable maximum separation between game logic and data presentation.



The benefits of this software architectural pattern are as follows:

1. Separating model and view makes the code cleaner and easier for debugging.
2. The model can still be reused if we want to make the same game in another microcontroller, which facilitates long-term development of the project.

This pattern is suitable for our project since:

1. Making the code clean is essential for game development and debugging.
2. There are at least 2 different views in this project, a view for game and another view for debugging.
3. One of us is good at logic design, while another one is good at controlling hardware components. A good division of work can be conducted.

### Variables

We initialized all the variables as **global** and extern for the consistent state in every sub-routine. Variables are divided into different purpose as follows:

```
/* timer variables */
u8 task1HeartBeat;
int count = 0;

/* state variables */
int state = 0; // 0: start page, 1: game page, 2: end page
int diff = 0; int score = 0; int skill = 2; int life = 3;
int ruleDisplayed = 0; int hintDisplayed = 0;
int input[6]; int solved = 1;
int rocketMove = 0; int refreshPending = 0;

/* equation definition variables */
int left, mid, right, opt1, opt2, ans;

/* display variables */
u16 color = 0x001F; u16 bgcolor = 0xFFE0;
int player_pos = 112;
int block_pos = 0; int block_end = 0; int input_pos = 0;
int rocket_x = 112; int rocket_y = 116;
int ans_defaultX, ans_actualX, ans_maxX;
```

## Main Program Flow

The main function is used to check the current state, input, and call suitable functions per 100 ms.

```
int main(void){
    Game_Init();
    while(1){
        if (task1HeartBeat >= 10){

            task1HeartBeat = 0;
            if(count % 3 == 2){
                IERG3810_DS0_Off();
                IERG3810_DS1_Off();
                IERG3810_Buzzer_Off();
            }

            if (state == 0){
                if (count == 0){
                    USART_print(2, "Game Start ->");
                    printStartScreen();
                }
                if (count % 2 == 0){
                    checkState0KeyInput();
                }
            }

            if (checkKeyboardInput()) continue;

            if (state == 1){
                if (count >= 100){
                    count = 0; // reset timer count
                    rocketMove = 0; // stop rocket movement
                    if (checkTimeout()) continue;
                }
                printPlayerIcon();
                checkState1KeyInput();
                printMainScreen();
                checkAns();
            }

            if (state == 2){
                if (count == 0){
                    USART_print(2, " Game Over! ");
                    printEndGame();
                }
                checkState2KeyInput();
            }
            count++;
        }
    }
}
```

## Subroutines

### calculate.c

This is the source file for game model. The following subroutines hold the game logic. It does all the functions that we need to start a new game.

```
// get the number of digit given an integer, return negative value if the integer is negative
int getNumOfDigit(int num){

}

// get the first digit given an integer, return -1 if the integer has only 1 digit
int getFirstDigit(int ans){

}

// perform arithmetic operation given an operator and 2 operands
int doOpt(int left, int right, int opt){

}

// generate a new equation
void genEqt(){

}
```

### game.c

This file includes the main operation of the game. For example, at first, we initialize the setup of all the peripherals. Then, the game needs to generate question block for player to solve. The interrupt check for player input. If it is wrong, clear input. Else, next question. The player may also want to reset game when playing or restart the game after they game over.

```
// initialize hardware components for the game
void Game_Init(){

}

// initialize a question
void Question_Init(){

}

// clear input
void clearInput(){

}

// restart game without back to the start screen
void restartGame(){

}

// restart game and back to the start screen
void resetGame(){

}
```

## input.c

This file is responsible for checking player input. The game needs to check player input in different state. For example, check what player answer, check if the answer is correct and check the timeout of the game.

```
// check timeout of the game
int checkTimeout() {

// check user keyboard input
int checkKeyboardInput() {

// check user key input when state = 0
void checkState0KeyInput() {

// check user key input when state = 1
void checkState1KeyInput() {

// check user key input when state = 2
void checkState2KeyInput() {

// check user answer
void checkAns() {
```

## display.c

This file stores all the sub-routines that for the display on TFTLCD screen. There are 12 things need to be print, for example start screen, game rule, player icon, etc. We divide the printing of the equation into 3 parts: operator, numbers and combination of operator and numbers.

```
// clean whole game screen
void cleanScreen() {
// print out start screen
void printStartScreen() {
// print out game rule
void printRule() {
// print out an integer, noBg: print without background, hide: replace every digit by '_'
int printInt(int num, int x, int y, int noBg, int hide) {
// print out an operator
int printOpt(int opt, int x, int y) {
// print out an equation
void printEqt(int x, int y) {
// print out upper panel
void printUpperPanel() {
// print out main screen
void printMainScreen() {
// print out player icon
void printPlayerIcon() {
// print out lower panel
void printLowerPanel() {
// print end game screen
void printEndGame() {
// print out hint
int printHint() {
// print out stop screen
void printStop() {
```

## display\_USART.c

This file includes all the operation regarding USART. The USART is mainly used for debugging. For example, when player open the game, the USART screen should show “Game Start ->”. It clearly describes the state of game, and even captures the player movement.

```
// print out an integer using USART
void printInt USART(int num) {
// print out an operator using USART
void printOpt USART(int opt) {
// print out an equation using USART
void printEqt USART() {
```

## Other File SFONT.H

```
#ifndef __SFONT_H
#define __SFONT_H

const unsigned char special_1616[3][32]={
{0x00,0x00,0x03,0x00,0x00,0x00,0x44,0x10,0x02,0x02,0x22,0x02,0x30,0x02,0x10,0x22,0x02,0x02,0x10,0x02,0x00,0x44,0x04,0x00,0x00,0x00}, /* Ditto, 1 */
{0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x0f,0x00,0x1f,0x0c,0x3f,0x02,0x79,0x04,0x79,0x04,0x3f,0x02,0x1f,0x0c,0x0f,0x00,0x00,0x00,0x00,0x00}, /* Rocket, 2 */
{0x00,0x00,0x0f,0x00,0x1f,0x00,0x3f,0x0c,0x3f,0x00,0x1f,0x0f,0x0f,0x07,0x0c,0x07,0x0c,0x0f,0x0f,0x1f,0x0f,0x3f,0x0c,0x3f,0x00,0x00,0x00}, /* Heart, 3 */
};

#endif
```

We drew three logos about the game which are heart for showing the lives player has, Ditto for showing the player position and Rocket for animation that player answer has submitted.

## VII. DIFFICULTIES

There are three main difficulties in this project.

First of all, the **technical issue**. We found that making animation is more difficult than we think. Initially, we think using timer and add a certain value in y-position can create a rocket-launching animation. However, a line is printed instead because we forgot to clear up the previous image. Therefore, a series of rocket logo is printed along y-direction. Fortunately, we solved the issue after knowing the reason.

Secondly, the **collaboration issue**. We argued for how to implement USART function. One of us wanted to use in debug, and other one wanted to use in showing instructions of the game. However, we reached consensus and implemented it to show debugging messages.

Finally, the **code version issue**. As it is hard to edit the program code together, sometimes there are some version conflicts. However, we found to use Git to do the version control. We made many branches for different sub-routine like “Display”, “Calculation” and “USART”. After all the work is done, we merge to the main branch. After that, the version conflict problem is resolved.

## VIII. SUMMARY

Throughout this mini project, we implemented what we learnt for the past few labs. We can make a mini game based on the features of STM32 including TFTLCD, LED, button, buzzer, USART. We also tried to improve the game day by day, for example we transit from polling to interrupt and use timer instead of Delay counter. We understand the principle behind these circuits. We also learnt how divide job duties like who to design the interface and who to design the game structure. Although sometimes we argue, soon we will reach consensus again. All in all, in this course, other than the knowledge behind STM32 board, we learn to collaborate and unleash our creativity.

## IX. DIVISION OF WORK

Game Design	KEI Yat-long and LI Hong Man
Game Logic and Control	KEI Yat-long
Game Presentation	LI Hong Man
Game Decoration	KEI Yat-long
Project Report	KEI Yat-long and LI Hong Man

## X. REFERENCES

STMicroelectronics. (2018). RM0008 Reference manual (Rev. 20). Geneva, Switzerland: Author.

ARM. (2007). Technical Reference Manual (Rev. r1p1). Cambridge, England: Author