ボタンリアクションゲーム 仕様書

1. システム構成

本システムは、Raspberry Pi、AQM0802A LCD ディスプレイ、タクトスイッチ 4 個を用いて構築される反応速度ゲームである。
LCD は I2C 接続により Raspberry Pi と通信し、タクトスイッチは GPIO22～25 に接続されている。

2. 使用技術と制御方式組み込み開発環境の概要

以下の機器とソフトウェアを使用して組み込みシステムを開発した。

機器

- **AQM0802A LCD ディスプレイ：**ゲームのスコアやメッセージを表示するために使用した。
- **Raspberry Pi：**組み込みシステムのホストとして使用した。
- **タクトスイッチ (GPIO 22, 23, 24, 25)**：ユーザー入力を受け取るために使用した。

ソフトウェア

- Linux: Raspberry Pi 上のオペレーティングシステム。
- GCC: プログラムをコンパイルするために使用した。
- **I2C デバイスドライバ**：LCD ディスプレイと通信するために使用した。

役割と手順

- **役割**：各コンポーネントの役割を明確に定義し、LCD ディスプレイは出力表示、タクトスイッチはユーザー入力、Raspberry Pi は制御とデータ処理を担当した。
- **手順**：以下の手順で開発を進めた。

  ① 開発環境のセットアップ
  ② 各ハードウェアコンポーネントの動作確認
  ③ プログラムの実装とデバッグ
  ④ システム全体の統合とテスト

3. 動作の概要

本システムは、複数のプログラムコンポーネントから構成されている。各コンポーネントがどのように連携しているかを以下に示す。

コンポーネント

・**入力系**：タクトスイッチ (GPIO 22, 23, 24, 25) によりユーザーの入力を取得する。
・**出力系**：AQM0802A LCD ディスプレイによりスコアやメッセージを表示する。

連携

タクトスイッチの入力に応じて、LCD ディスプレイに表示を更新する。ゲームの開始前にコントラスト設定を行い、ユーザーが適切なコントラストを設定した後にゲームが開始される。ゲーム中は、ユーザーの入力に応じてスコアが増減し、30 秒間で得点を競う。

1. システムの詳細

以下に、自作したプログラムとその説明を示す。

ソースコード

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <fcntl.h>
#include <errno.h>
#include <sys/ioctl.h>
#include <linux/i2c-dev.h>
#include <time.h>

#define I2C_ADDR 0x3E // AQM0802A の I2C アドレス（7bit 表記で 0x3E、8bit 表記では 0x7C）
#define LCD_WIDTH 8 // AQM0802A の幅
#define LCD_CHR 1
#define LCD_CMD 0

#define LCD_LINE_1 0x80
#define LCD_LINE_2 0xC0

#define E_PULSE 500
```

```c
#define E_DELAY 500

#define BUTTON_START 22 // 最初のタクトスイッチの GPIO 番号
#define BUTTON_END 25   // 最後のタクトスイッチの GPIO 番号
#define BUTTON_COUNT (BUTTON_END - BUTTON_START + 1)

// 表示位置とボタンの対応
int positions[4][2] = {
    {0, 0}, // 1 行目左端
    {1, 0}, // 2 行目左端
    {1, 7}, // 2 行目右端
    {0, 7}  // 1 行目右端
};

int buttons[4] = {22, 23, 24, 25}; // 対応するボタン

int lcd_cmdwrite(int fd, unsigned char dat);
int lcd_datawrite(int fd, char dat[]);
void initLCD(int fd);
int location(int fd, int y, int x);
int clear(int fd);
void gpio_export(int pin);
void gpio_unexport(int pin);
void gpio_set_direction(int pin, int direction);
int gpio_read(int pin);
void set_contrast(int fd, unsigned char contrast);
void contrast_setup(int fd);

void gpio_export(int pin) {
    char buffer[3];
    int len;
    int fd = open("/sys/class/gpio/export", O_WRONLY);
    if (fd < 0) {
        perror("gpio/export");
        exit(1);
    }
    len = snprintf(buffer, sizeof(buffer), "%d", pin);
    write(fd, buffer, len);
    close(fd);
}

void gpio_unexport(int pin) {
    char buffer[3];
    int len;
    int fd = open("/sys/class/gpio/unexport", O_WRONLY);
```

```
    if (fd < 0) {
        perror("gpio/unexport");
        exit(1);
    }
    len = snprintf(buffer, sizeof(buffer), "%d", pin);
    write(fd, buffer, len);
    close(fd);
}

void gpio_set_direction(int pin, int direction) {
    char path[35];
    snprintf(path, sizeof(path), "/sys/class/gpio/gpio%d/direction", pin);
    int fd = open(path, O_WRONLY);
    if (fd < 0) {
        perror("gpio/direction");
        exit(1);
    }
    if (direction == 0) {
        write(fd, "in", 2);
    } else {
        write(fd, "out", 3);
    }
    close(fd);
}

int gpio_read(int pin) {
    char path[30];
    char value_str[3];
    snprintf(path, sizeof(path), "/sys/class/gpio/gpio%d/value", pin);
    int fd = open(path, O_RDONLY);
    if (fd < 0) {
        perror("gpio/value");
        exit(1);
    }
    if (read(fd, value_str, 3) < 0) {
        perror("gpio/read");
        exit(1);
    }
    close(fd);
    return atoi(value_str);
}

void countdown(int fd) {
    for (int i = 3; i > 0; i--) {
        clear(fd);
```

```
        location(fd, 0, 3);
        char count_str[2];
        sprintf(count_str, "%d", i);
        lcd_datawrite(fd, count_str);
        usleep(1000000); // 1 秒待機
    }
    clear(fd);
    location(fd, 0, 0);
    lcd_datawrite(fd, "Start!");
    usleep(1000000); // 1 秒待機
    clear(fd);
}

void contrast_setup(int fd) {
    clear(fd);
    location(fd, 0, 0);
    lcd_datawrite(fd, "contrast");
    location(fd, 1, 0);
    lcd_datawrite(fd, "L:b, H:r");

    int contrast_set = 0;
    while (!contrast_set) {
        if (gpio_read(22) == 0) {
            set_contrast(fd, 0x15); // コントラストを 0x15 に設定
            contrast_set = 1;
        } else if (gpio_read(23) == 0) {
            set_contrast(fd, 0x22); // コントラストを 0x22 に設定
            contrast_set = 1;
        }
    }

    clear(fd);
    location(fd, 0, 0);
    lcd_datawrite(fd, "Contrast");
    location(fd, 1, 0);
    lcd_datawrite(fd, "Set!");
    usleep(1000000); // 1 秒待機
    clear(fd);
}

int main() {
    int fd;
    int score = 0;
    int last_pos_index = -1; // 前回の表示位置を記憶
    time_t start_time, current_time;
```

```
    // GPIO の設定
    for (int pin = BUTTON_START; pin <= BUTTON_END; pin++) {
        gpio_export(pin);
        gpio_set_direction(pin, 0); // 0 は入力
    }

    // LCD 初期化
    if ((fd = open("/dev/i2c-1", O_RDWR)) < 0) {
        perror("Failed to open the i2c bus");
        return 1;
    }
    if (ioctl(fd, I2C_SLAVE, I2C_ADDR) < 0) {
        perror("Failed to acquire bus access and/or talk to slave");
        return 1;
    }
    initLCD(fd);
    clear(fd);

    // コントラスト設定
    contrast_setup(fd); // コントラスト設定の確認と変更

    // カウントダウン表示
    countdown(fd);

    srand(time(NULL));
    time(&start_time);

    while (1) {
        time(&current_time);
        if (difftime(current_time, start_time) > 30) break;

        int pos_index;
        do {
            pos_index = rand() % 4; // 4 つの表示位置からランダムに選択
        } while (pos_index == last_pos_index); // 直前の位置と同じ場合は再選択

        last_pos_index = pos_index;

        int y = positions[pos_index][0];
        int x = positions[pos_index][1];
        int correct_button = buttons[pos_index]; // 表示位置に対応するボタン

        clear(fd);
        location(fd, y, x);
```

```
        lcd_datawrite(fd, "*");

        int button_pressed = 0;
        while (!button_pressed) {
            for (int pin = BUTTON_START; pin <= BUTTON_END; pin++) {
                if (gpio_read(pin) == 0) {
                    if (pin == correct_button) {
                        score++;
                        printf("Correct button pressed. Score: %d\n", score);
                    } else {
                        score--;
                        printf("Wrong button pressed. Score: %d\n", score);
                    }
                    button_pressed = 1;
                    break;
                }
            }
        }
        usleep(500000); // 0.5 秒待機
    }

    clear(fd);
    location(fd, 0, 0);
    lcd_datawrite(fd, "Score:");
    char score_str[16];
    sprintf(score_str, "%d", score);
    int len = strlen(score_str);
    int pos = (LCD_WIDTH - len) / 2;
    location(fd, 1, pos); // 2 行目に表示する
    lcd_datawrite(fd, score_str);

    close(fd);
    for (int pin = BUTTON_START; pin <= BUTTON_END; pin++) {
        gpio_unexport(pin);
    }
    return 0;
}

int lcd_cmdwrite(int fd, unsigned char dat) {
    unsigned char buff[2];
    buff[0] = 0x00; // Co = 0, RS = 0
    buff[1] = dat;
    return write(fd, buff, 2);
}
```

```c
int lcd_datawrite(int fd, char dat[]) {
    int len;
    char buff[100];

    len = strlen(dat);
    if (len > 99) {
        printf("too long string\n");
        exit(1);
    }
    memcpy(buff + 1, dat, len);
    buff[0] = 0x40; // Co = 0, RS = 1
    return write(fd, buff, len + 1);
}

void initLCD(int fd) {
    usleep(50000); // 50ms wait

    lcd_cmdwrite(fd, 0x38); // Function set: 8-bit mode
    lcd_cmdwrite(fd, 0x39); // Function set: 8-bit mode + instruction table 1
    lcd_cmdwrite(fd, 0x14); // Internal OSC frequency
    lcd_cmdwrite(fd, 0x70); // Contrast set
    lcd_cmdwrite(fd, 0x56); // Power/ICON/Contrast control
    lcd_cmdwrite(fd, 0x6C); // Follower control
    usleep(200000);         // 200ms wait

    lcd_cmdwrite(fd, 0x38); // Function set: 8-bit mode
    lcd_cmdwrite(fd, 0x0C); // Display ON
    lcd_cmdwrite(fd, 0x01); // Clear display
    usleep(2000);           // 2ms wait
}

int location(int fd, int y, int x) {
    int cmd = 0x80 + y * 0x40 + x;
    return lcd_cmdwrite(fd, cmd);
}

int clear(int fd) {
    int val = lcd_cmdwrite(fd, 0x01);
    usleep(2000); // 2ms wait
    return val;
}

void set_contrast(int fd, unsigned char contrast) {
    lcd_cmdwrite(fd, 0x39); // Function set: 8-bit mode + instruction table 1
    lcd_cmdwrite(fd, 0x70 | (contrast & 0x0F)); // コントラスト設定
```

```
    lcd_cmdwrite(fd, 0x5C | ((contrast >> 4) & 0x03)); // Power/ICON/Contrast
control
    lcd_cmdwrite(fd, 0x38); // Function set: 8-bit mode
}
```

プログラムの説明

本プログラムは、Raspberry Pi 上で動作し、タクトスイッチの入力に応じて LCD ディスプレイに表示を行う組み込みシステムである。

標準ライブラリのインクルード

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <fcntl.h>
#include <errno.h>
#include <sys/ioctl.h>
#include <linux/i2c-dev.h>
#include <time.h>
```

- **stdio.h**：標準入出力ライブラリ。**printf** や **sprintf** などの関数を使用。
- **stdlib.h**：標準ライブラリ。**malloc** や **free**、**exit** などの関数を使用。
- **string.h**：文字列操作ライブラリ。**strlen** や **memcpy** などの関数を使用。
- **unistd.h**：UNIX 標準ライブラリ。**usleep** や **close** などの関数を使用。
- **fcntl.h**：ファイル制御ライブラリ。**open** 関数を使用。
- **errno.h**：エラー処理ライブラリ。エラー番号を取得するために使用。
- **sys/ioctl.h**：入出力制御ライブラリ。**ioctl** 関数を使用。
- **linux/i2c-dev.h**：I2C デバイスドライバ。I2C 通信を行うために使用。
- **time.h**：時間管理ライブラリ。**time** 関数を使用。

マクロ定義

```
#define I2C_ADDR 0x3E // AQM0802A の I2C アドレス（7bit 表記で 0x3E、8bit 表記では 0x7C）
#define LCD_WIDTH 8 // AQM0802A の幅
#define LCD_CHR 1
#define LCD_CMD 0

#define LCD_LINE_1 0x80
#define LCD_LINE_2 0xC0

#define E_PULSE 500
```

```
#define E_DELAY 500

#define BUTTON_START 22 // 最初のタクトスイッチの GPIO 番号
#define BUTTON_END 25   // 最後のタクトスイッチの GPIO 番号
#define BUTTON_COUNT (BUTTON_END - BUTTON_START + 1)
```

- **I2C_ADDR:** AQM0802A LCD ディスプレイの I2C アドレス。7 ビット表記で 0x3E。
- **LCD_WIDTH**：LCD の幅。8 文字。
- **LCD_CHR**：LCD にデータを書き込む際のフラグ。データモード。
- **LCD_CMD**：LCD にコマンドを書き込む際のフラグ。コマンドモード。
- **LCD_LINE_1**：LCD の 1 行目のアドレス。
- **LCD_LINE_2**：LCD の 2 行目のアドレス。
- **E_PULSE**：イネーブルパルスの長さ。
- **E_DELAY**：イネーブル遅延。
- **BUTTON_START**：最初のタクトスイッチの GPIO 番号。
- **BUTTON_END**：最後のタクトスイッチの GPIO 番号。
- **BUTTON_COUNT**：使用するタクトスイッチの数。

表示位置とボタンの対応

```
// 表示位置とボタンの対応
int positions[4][2] = {
    {0, 0}, // 1 行目左端
    {1, 0}, // 2 行目左端
    {1, 7}, // 2 行目右端
    {0, 7}  // 1 行目右端
};

int buttons[4] = {22, 23, 24, 25}; // 対応するボタン
```

- **positions**：LCD の表示位置を行と列で定義。4 つの位置（1 行目左端、2 行目左端、2 行目右端、1 行目右端）。
- **Buttons**：タクトスイッチの GPIO 番号。各位置に対応するボタン。

GPIO 関連関数

```
void gpio_export(int pin) {
    char buffer[3];
    int len;
    int fd = open("/sys/class/gpio/export", O_WRONLY);
    if (fd < 0) {
        perror("gpio/export");
        exit(1);
```

```
    }
    len = snprintf(buffer, sizeof(buffer), "%d", pin);
    write(fd, buffer, len);
    close(fd);
}
```

・**gpio_export 関数**：指定した GPIO ピンを有効にするために、システムファイルに書き込みを行う。

・**引数**

pin：有効にする GPIO ピンの番号。

・**処理**

・**/sys/class/gpio/export** ファイルを開く。

・**snprintf** 関数を使用して、GPIO ピン番号を文字列に変換。

・write 関数を使用して、変換された文字列を**/sys/class/gpio/export** ファイルに書き込み、GPIO ピンを有効にする。

・ファイルを閉じる。

```
void gpio_unexport(int pin) {
    char buffer[3];
    int len;
    int fd = open("/sys/class/gpio/unexport", O_WRONLY);
    if (fd < 0) {
        perror("gpio/unexport");
        exit(1);
    }
    len = snprintf(buffer, sizeof(buffer), "%d", pin);
    write(fd, buffer, len);
    close(fd);
}
```

・**gpio_unexport 関数**：指定した GPIO ピンを無効にするために、システムファイルに書き込みを行う。

・**引数**

pin：無効にする GPIO ピンの番号。

・**処理**

・**/sys/class/gpio/unexport** ファイルを開く。

・**snprintf** 関数を使用して、GPIO ピン番号を文字列に変換。

・**write** 関数を使用して、変換された文字列を**/sys/class/gpio/unexport** ファイルに書き込み、GPIO ピンを無効にする。

・ファイルを閉じる。

```c
void gpio_set_direction(int pin, int direction) {
    char path[35];
    snprintf(path, sizeof(path), "/sys/class/gpio/gpio%d/direction", pin);
    int fd = open(path, O_WRONLY);
    if (fd < 0) {
        perror("gpio/direction");
        exit(1);
    }
    if (direction == 0) {
        write(fd, "in", 2);
    } else {
        write(fd, "out", 3);
    }
    close(fd);
}
```

・**gpio_set_direction 関数**：指定した GPIO ピンの方向を設定する。方向は入力（"in"）または出力（"out"）を指定できる

・引数

・**pin**：設定する GPIO ピンの番号。

・**direction**：ピンの方向（0 は入力、1 は出力）。

・処理

・GPIO ピンの方向を設定するファイルパスを生成。

・パスに基づいてファイルを開く。

・**write** 関数を使用して、入力方向の場合は「in」、出力方向の場合は「out」を書き込み、ピンの方向を設定。

・ファイルを閉じる。

```c
int gpio_read(int pin) {
    char path[30];
    char value_str[3];
    snprintf(path, sizeof(path), "/sys/class/gpio/gpio%d/value", pin);
    int fd = open(path, O_RDONLY);
    if (fd < 0) {
        perror("gpio/value");
        exit(1);
    }
    if (read(fd, value_str, 3) < 0) {
        perror("gpio/read");
        exit(1);
    }
    close(fd);
    return atoi(value_str);
```

```
}
```

・**gpio_read 関数**：指定した GPIO ピンの値を読み取る。値は 0（LOW）または 1（HIGH）である。

・**引数**

pin：読み取る GPIO ピンの番号。

・**処理**

・GPIO ピンの値を読み取るファイルパスを生成。

・パスに基づいてファイルを開く。

・**read** 関数を使用して、ピンの値を読み取る。

・読み取った値を整数に変換し、返す。

・**戻り値**：ピンの値（0 または 1）。

カウントダウン関数

```c
void countdown(int fd) {
    for (int i = 3; i > 0; i--) {
        clear(fd);
        location(fd, 0, 3);
        char count_str[2];
        sprintf(count_str, "%d", i);
        lcd_datawrite(fd, count_str);
        usleep(1000000); // 1 秒待機
    }
    clear(fd);
    location(fd, 0, 0);
    lcd_datawrite(fd, "Start!");
    usleep(1000000); // 1 秒待機
    clear(fd);
}
```

・**countdown 関数**：ゲーム開始前にカウントダウンを表示する。LCD に 3 から 1 までのカウントダウンを表示し、その後「Start!」を表示。

・**引数**

fd：I2C デバイスファイルディスクリプタ。

・**処理**

・カウントダウンを 3 から 1 まで表示し、各カウントごとに 1 秒待機。

・カウントダウンが終わったら「Start!」を表示し、1 秒待機。

・最後に LCD をクリア。

コントラスト設定関数

```c
void contrast_setup(int fd) {
```

```
    clear(fd);
    location(fd, 0, 0);
    lcd_datawrite(fd, "contrast");
    location(fd, 1, 0);
    lcd_datawrite(fd, "L:b, H:r");

    int contrast_set = 0;
    while (!contrast_set) {
        if (gpio_read(22) == 0) {
            set_contrast(fd, 0x15); // コントラストを 0x15 に設定
            contrast_set = 1;
        } else if (gpio_read(23) == 0) {
            set_contrast(fd, 0x22); // コントラストを 0x22 に設定
            contrast_set = 1;
        }
    }

    clear(fd);
    location(fd, 0, 0);
    lcd_datawrite(fd, "Contrast");
    location(fd, 1, 0);
    lcd_datawrite(fd, "Set!");
    usleep(1000000); // 1 秒待機
    clear(fd);
}
```

・**contrast_setup 関数**：ゲーム開始前にユーザーがコントラストを設定できるようにする。LCD に「contrast」と「L, H」を表示し、GPIO 22 番または 23 番を押すことでコントラストを設定。設定が完了すると、「Contrast」と「Set!」が表示される。

・引数

fd: I2C デバイスファイルディスクリプタ。

・処理

・LCD をクリアし、「contrast」と「L, H」を表示。

・GPIO 22 番または 23 番が押されるまでループ。

・GPIO 22 番が押された場合、コントラストを 0x15 に設定。

・GPIO 23 番が押された場合、コントラストを 0x22 に設定。

・設定が完了すると、「Contrast」と「Set!」を表示し、1 秒待機。

メイン関数

```
int main() {
    int fd;
```

```
    int score = 0;
    int last_pos_index = -1; // 前回の表示位置を記憶
    time_t start_time, current_time;

    // GPIO の設定
    for (int pin = BUTTON_START; pin <= BUTTON_END; pin++) {
        gpio_export(pin);
        gpio_set_direction(pin, 0); // 0 は入力
    }

    // LCD 初期化
    if ((fd = open("/dev/i2c-1", O_RDWR)) < 0) {
        perror("Failed to open the i2c bus");
        return 1;
    }
    if (ioctl(fd, I2C_SLAVE, I2C_ADDR) < 0) {
        perror("Failed to acquire bus access and/or talk to slave");
        return 1;
    }
    initLCD(fd);
    clear(fd);

    // コントラスト設定
    contrast_setup(fd); // コントラスト設定の確認と変更

    // カウントダウン表示
    countdown(fd);

    srand(time(NULL));
    time(&start_time);

    while (1) {
        time(&current_time);
        if (difftime(current_time, start_time) > 30) break;

        int pos_index;
        do {
            pos_index = rand() % 4; // 4 つの表示位置からランダムに選択
        } while (pos_index == last_pos_index); // 直前の位置と同じ場合は再選択

        last_pos_index = pos_index;

        int y = positions[pos_index][0];
        int x = positions[pos_index][1];
        int correct_button = buttons[pos_index]; // 表示位置に対応するボタン
```

```
        clear(fd);
        location(fd, y, x);
        lcd_datawrite(fd, "*");

        int button_pressed = 0;
        while (!button_pressed) {
            for (int pin = BUTTON_START; pin <= BUTTON_END; pin++) {
                if (gpio_read(pin) == 0) {
                    if (pin == correct_button) {
                        score++;
                        printf("Correct button pressed. Score: %d\n", score);
                    } else {
                        score--;
                        printf("Wrong button pressed. Score: %d\n", score);
                    }
                    button_pressed = 1;
                    break;
                }
            }
        }
        usleep(500000); // 0.5 秒待機
    }

    clear(fd);
    location(fd, 0, 0);
    lcd_datawrite(fd, "Score:");
    char score_str[16];
    sprintf(score_str, "%d", score);
    int len = strlen(score_str);
    int pos = (LCD_WIDTH - len) / 2;
    location(fd, 1, pos); // 2 行目に表示する
    lcd_datawrite(fd, score_str);

    close(fd);
    for (int pin = BUTTON_START; pin <= BUTTON_END; pin++) {
        gpio_unexport(pin);
    }
    return 0;
}
```

・**main 関数**：プログラムのエントリーポイント。

・GPIO ピンの設定とエクスポート。

・LCD の初期化とコントラスト設定。

・カウントダウン表示。

・ゲームループ：
  ・30 秒間ランダムな位置に「*」を表示。
  ・ユーザーの入力を監視し、正しいボタンが押された場合にスコアを増加、間違ったボタンが押された場合にスコアを減少。
・ゲーム終了後のスコア表示。
・GPIO ピンのアンエクスポート。

LCD コマンドとデータ送信関数

```
int lcd_cmdwrite(int fd, unsigned char dat) {
    unsigned char buff[2];
    buff[0] = 0x00; // Co = 0, RS = 0
    buff[1] = dat;
    return write(fd, buff, 2);
}
```

・**lcd_cmdwrite 関数**：LCD にコマンドを送信する。バッファの最初のバイトにコマンドフラグ、次のバイトにデータを格納し I2C バスに送信。

```
int lcd_datawrite(int fd, char dat[]) {
    int len;
    char buff[100];

    len = strlen(dat);
    if (len > 99) {
        printf("too long string\n");
        exit(1);
    }
    memcpy(buff + 1, dat, len);
    buff[0] = 0x40; // Co = 0, RS = 1
    return write(fd, buff, len + 1);
}
```

・**lcd_datawrite 関数**：LCD にデータを送信する。バッファの最初のバイトにデータフラグ、次のバイトからデータを格納し、I2C バスに送信。

LCD 初期化関数

```
void initLCD(int fd) {
    usleep(50000); // 50ms wait

    lcd_cmdwrite(fd, 0x38); // Function set: 8-bit mode
    lcd_cmdwrite(fd, 0x39); // Function set: 8-bit mode + instruction table 1
    lcd_cmdwrite(fd, 0x14); // Internal OSC frequency
    lcd_cmdwrite(fd, 0x70); // Contrast set
```

```
    lcd_cmdwrite(fd, 0x56); // Power/ICON/Contrast control
    lcd_cmdwrite(fd, 0x6C); // Follower control
    usleep(200000);         // 200ms wait

    lcd_cmdwrite(fd, 0x38); // Function set: 8-bit mode
    lcd_cmdwrite(fd, 0x0C); // Display ON
    lcd_cmdwrite(fd, 0x01); // Clear display
    usleep(2000);           // 2ms wait
}
```

・**initLCD 関数**：LCD の初期化を行う。各種設定コマンドを送信し、LCD を初期化状態にする。

カーソル位置指定関数

```
int location(int fd, int y, int x) {
    int cmd = 0x80 + y * 0x40 + x;
    return lcd_cmdwrite(fd, cmd);
}
```

・**location 関数**：LCD のカーソル位置を指定する。LCD の行と列を指定し、カーソルを移動する

クリア関数

```
int clear(int fd) {
    int val = lcd_cmdwrite(fd, 0x01);
    usleep(2000); // 2ms wait
    return val;
}
```

・**clear 関数**：LCD をクリアする。クリアコマンドを送信し、ディスプレイをリセットする。

コントラスト設定関数

```
void set_contrast(int fd, unsigned char contrast) {
    lcd_cmdwrite(fd, 0x39); // Function set: 8-bit mode + instruction table 1
    lcd_cmdwrite(fd, 0x70 | (contrast & 0x0F)); // コントラスト設定
    lcd_cmdwrite(fd, 0x5C | ((contrast >> 4) & 0x03)); // Power/ICON/Contrast
control
    lcd_cmdwrite(fd, 0x38); // Function set: 8-bit mode
}
```

・**set_contrast 関数**：LCD のコントラストを設定する。contrast 引数で指定した値に基づいてコントラストを調整する。

・引数

・**fd**：I2C デバイスファイルディスクリプタ。

・**contrast**：設定するコントラストの値（0x00 から 0x3F）。

・処理

・lcd_cmdwrite 関数を使用して、コントラスト設定のためのコマンドを LCD に送信。

・**0x39**：8 ビットモード + 拡張命令テーブル 1 を設定。

・**0x70 | (contrast & 0x0F)**：コントラストの下位 4 ビットを設定。

・**0x5C | ((contrast >> 4) & 0x03)**：コントラストの上位 2 ビットを設定し、パワー/アイコン制御を設定。

・**0x38**：8 ビットモードを設定。

4. 操作説明

ユーザーは初めにコントラスト設定を行う。

ユーザーがタクトスイッチを押して LCD にランダムに表示された「*」の位置に対応するボタンを時間内になるべく多く押し、点数を競うゲーム

・**コントラスト設定**

LCD に「Contrast」「L： b (brown), H：r(red)」と表示され、低コントラストを指定する場合は GPIO22(茶)ボタン、高コントラストを指定する場合は GPIO23(赤)ボタンを押す。ボタンを押した後、「Contrast Set!」と表示される。

図１：高コントラスト

図２：低コントラスト

・**カウントダウン**

コントラスト設定が終了してすぐにカウントダウンがスタート

「３」「２」「１」「Start！」が順番に表示される

図３：「３」

図４：「２」

図５：「１」

図６：「Start!」

・位置と対応するボタン

・上段左端：GPIO22(茶)ボタン
・下段左端：GPIO23(赤)ボタン
・下段右端：GPIO24(橙)ボタン
・上段右端：GPIO25(黄)ボタン

図 7：上段左端

図 8：下段左端

図 9：下段右端

図 10：上段右端

・点数の仕様

押したボタンが表示位置に合っていると+1 点、間違えると-1 点
正答するとエディタに「Correct button pressed. Score: “現在の点数”」、
誤答すると「Wrong button pressed. Score: “現在の点数”」と表示される。
終了した際、LCD には「Score:」「“最終的な点数”」が表示される。