**<화.목요일 4조>**

# 전기전자심화설계및실습
# <텀프로젝트-최종>
## (미니게임 – 오셀로)


**담당교수 : 조용범 교수님/유위 교수님**

**실험날짜 : 2020.12.18**

**조 : 화.목요일 4조**

**이름 : 201611334 김한리**

**201611344 박범수**

**201611396 진시민**

# 1. Title

* 텀 프로젝트 최종 보고서 (오셀로 구현)

# 2. Name

201611334 김한리
201611344 박범수
201611396 진시민

# 3. Abstract

* SV210 보드를 이용한 오셀로 게임을 구현합니다.
(1) Camera로 플레이어를 등록합니다.
(2) Dotmatrix로 현재 어떤 플레이어의 순서인지 알려줍니다.
(3) Buzzer로 해당 위치를 통해 돌을 뒤집을 수 없는 경우를 알려줍니다.
(4) TouchLCD로 오셀로 보드 출력과 돌의 위치를 입력받을 뿐아니라 전반적인 오셀로를 진행합니다.
* 게임이 끝날 경우, 승패 또는 무승부를 출력하고 다시 게임 시작으로 돌아갑니다.

# 4. Background

## [1] 디바이스 드라이버

* 시스템이 지원하는 하드웨어를 응용 프로그램에서 사용할 수 있도록 커널에서 제공하는 라이브러리로, 응용 프로그램이 커널에게 자원 사용을 요청하고 커널은 이러한 요청에 따라 시스템을 관리한다. 모듈 초기화 함수에서는 디바이스 드라이버가 처리해야 할 하드웨어 검출 및 검출된 하드웨어를 응용 프로그램에서 사용할 수 있도록 초기화 작업을 한다. module_init 에서는 처리하고자 하는 하드웨어 검출과 디바이스 드라이버 등록 그리고 디바이스 드라이버가 응용 프로그램에서 사용할 수 있는 환경(내부 구조체의 메모리 할당) 등을 처리한다. 반대로 module_exit 에서는 디바이스 드라이버의 해제 및 할당된 메모리 해제, 하드웨어 제거에 따른 처리를 담당한다.

## [2] 디바이스 파일

* 리눅스는 시스템의 모든 자원을 파일 형식으로 표현하기 때문에 디바이스 파일 하나하나가 실질적인 하드웨어를 표현한다. 디바이스 파일은 하드웨어 정보를 제공하며 정보는 디바이스 타입 정보, 주 번호, 부 번호로 나뉜다. 주 번호는 디바이스를 구분하기 위한 ID 이고, 부 번호는 같은 종류의 디바이스 중 하나를 구분하기 위해 사용된다. 디바이스 드라이버의 타입은 문자/블록/네트워크로 나뉘며 문자는 시리얼, 콘솔, 키보드 등이, 블록엔 하드디스크, 램디스크, 네트워크에는 이더넷 PPP 등이 해당된다. 디바이스 파일은 create()를 사용하지 않고 mknod 를 이용해 /dev/ 아래에 생성된다.

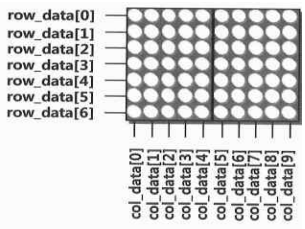
## [3] 모듈

리눅스 커널이 동작중인 상태에서 디바이스 드라이버를 동적으로 추가하거나 제거할 수 있게 하는 개념으로 개발 시간을 단축시킬 뿐만 아니라 필요없는 기능은 커널에서 제외함으로써 자원을 효율적으로 다룰 수 있게 한다. 유저 영역에서 동작하는 일반 프로그램과 달리 커널 모듈 프로그램은 커널 영역에서 동작하며 커널이 export 해준 함수만 사용할 수 있다. 모듈로 만들어진 디바이스 드라이버는 insmod, rmmod, lsmod 등의 유틸리티를 통해 커널에 링크된다.

## [4] SV_210

Cortex-A8 Core 를 내장한 삼성전자의 1GHz S5PV210 MCU 를 탑재하였고 Linux 2.6.35 커널의 전체 BSP 소스, Windows CE 6.0, 안드로이드(Android) 2.3, 4.0 ICS BSP 소스 코드와 모든 장치 드라이버 소스 또한 포함되어 있다. HDMI, TV OUT, CMOS Camera, Wi-Fi, GPS, WCDMA 등의 다양한 주변 장치의 드라이버 및 예제 코드를 지원하고 있다.

## [5] Dot Matrix




* LED 를 격자 모양으로 배열한 소자를 Dot matrix 라고 한다. Dot matrix controller 는 16bit 로 구성된 2 개의 register, Dot_Col_Reg 와 Dot_Row_Reg 에 의해 제어된다. 이 register 들은 read 와 write 모두 가능하도록 설계되어 있으며 7 행 10 열로 구성된 DOT matrix 의 행과 열을 선택하여 LED 를 선택할 수 있다. Dot_Col_Reg 의 bit0~9 에 '0'~'1'을 기록하여 열을 제어하고, Dot_Row_Reg 의 bit0~6 에 '0'또는 '1'을 기록함으로써 행을 제어한다. Dot_Col_Reg 와 Dot_Row_Reg 은 모두 active high 로 동작해 '0'인 경우 해제되며 '1'인 경우 선택된다. Dot_Row_Reg 의 15 번째 bit 는 '0'인 경우 col 또는 row 의 값에 상관없이 OFF 되며 '1'인 경우 인가되는 신호에 따라 ON 이 되는 DOT matrix enable bit 이다. 물리주소는 0x8800_0042, 0x8800_0040 에 해당하며 주번호는 244 를 사용했다.

## [6] TouchLCD

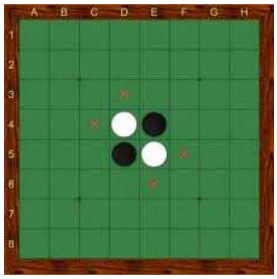
크기는 800*480 으로, touchLCD 는 크기가 커 일일이 접근하기 어렵기 때문에 mmap 을 이용해 임의의 주소에 메모리 매핑을 한 후 접근한다. touch 가 발생하면 /dev/input/event2 에서 인터럽트를 읽어와 범위에 따라 x_detected 와 y_detected 에 좌표를 x, y 축을 기준으로 -1~8 의 값을 return 한다.

## [7] Othello 규칙

(1) 게임 시작

* 시작으로는 중앙에 흑색, 백색 돌을 각각 2개씩 교차해서 놓습니다.

(2) 진행 과정



* 돌을 놓을 때, 자신의 돌과 둘려는 곳 사이에 상대편의 돌이 있어야 놓을 수 있습니다. 둘 수 있는 곳에 자신의 돌을 놓으면, 그 사이에 있는 상대방의 돌이 자신의 색으로 바뀝니다.

(3) 턴 넘어가는 방식

* 자신의 돌을 놓아서 상대방의 돌을 바꿉니다. OR 자신의 돌을 둘 곳이 없습니다.

(4) 종료

* 판에 있는 모든 칸에 돌이 놓입니다. OR * 양쪽 다 돌을 둘 수 있는 곳이 없습니다.
* 종료될 때, 양측 돌 중에서 많은 플레이어가 승리하게 됩니다.
* 만약, 양측 돌의 수가 똑같으면 무승부가 됩니다.

## [8] 비트맵 헤더 구성

BITAMPINFOHEADER구조체

| Field Name | Size | Description |
|---|---|---|
| biSize | 4 | 헤더 크기(최소 40bytes) |
| biWidth | 4 | 이미지 폭 |
| biHeight | 4 | 이미지 높이 |
| biPlanes | 2 | 현재 지원값은 1입니다. |
| biBitCount | 2 | 비트수 1,4,8,16,24,32 |
| biCompression | 4 | 압축타입 : BI_RGB(0),BI_RLE8(1),BI_RLE4(2),BI_BITFIELDS(3) |
| biSizeImage | 4 | 이미지 크기 |
| biXPelsPerMeter | 4 | 미터당 픽셀수 x축 |
| biYPelsPerMeter | 4 | 미터당 픽셀수 y축 |
| biClrUsed | 4 | 실질적으로 사용될 컬러맵의 엔트리수 |
| biClrImportant | 4 | 주로 사용되는 컬러수 |

BITMAPCOREHEADER구조체

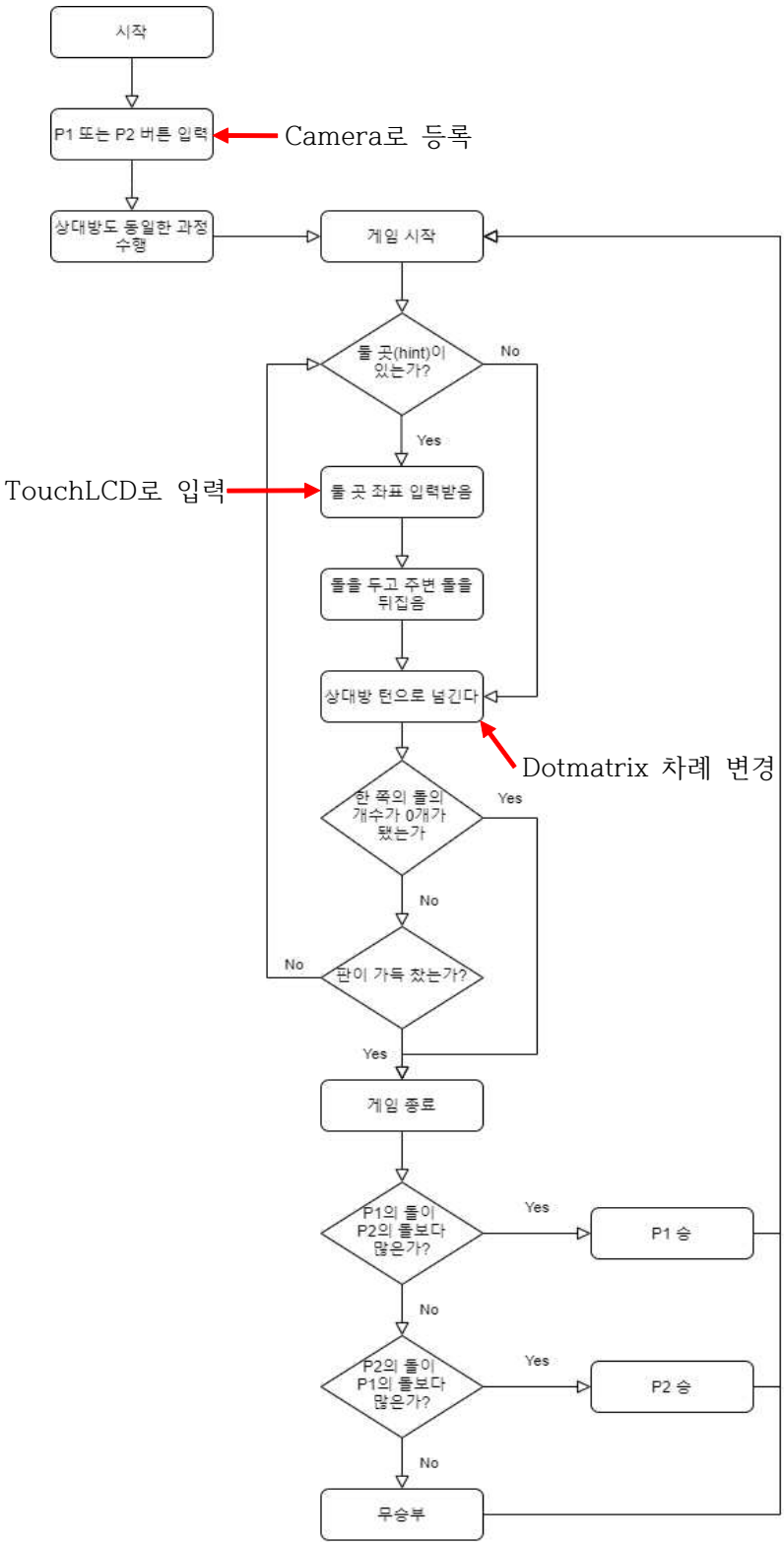
| Field Name | Size | Description |
|---|---|---|
| bcSize | 4 | 헤더 크기(12bytes) |
| bcWidth | 2 | 이미지 폭 |
| bcHeight | 2 | 이미지 높이 |
| bcPlanes | 2 | 현재 지원값은 1입니다. |
| bcBitCount | 2 | 비트수 1,4,8,24 |

* 다음과 같이 비트맵 헤더는 총 54bits를 차지하게 되며, 이후부터 이미지를 구성하는 데이터를 구성한다. Bitmap은 각 이미지 크기에 따라 4, 8, 16, 24, 32bits로 나뉩니다.

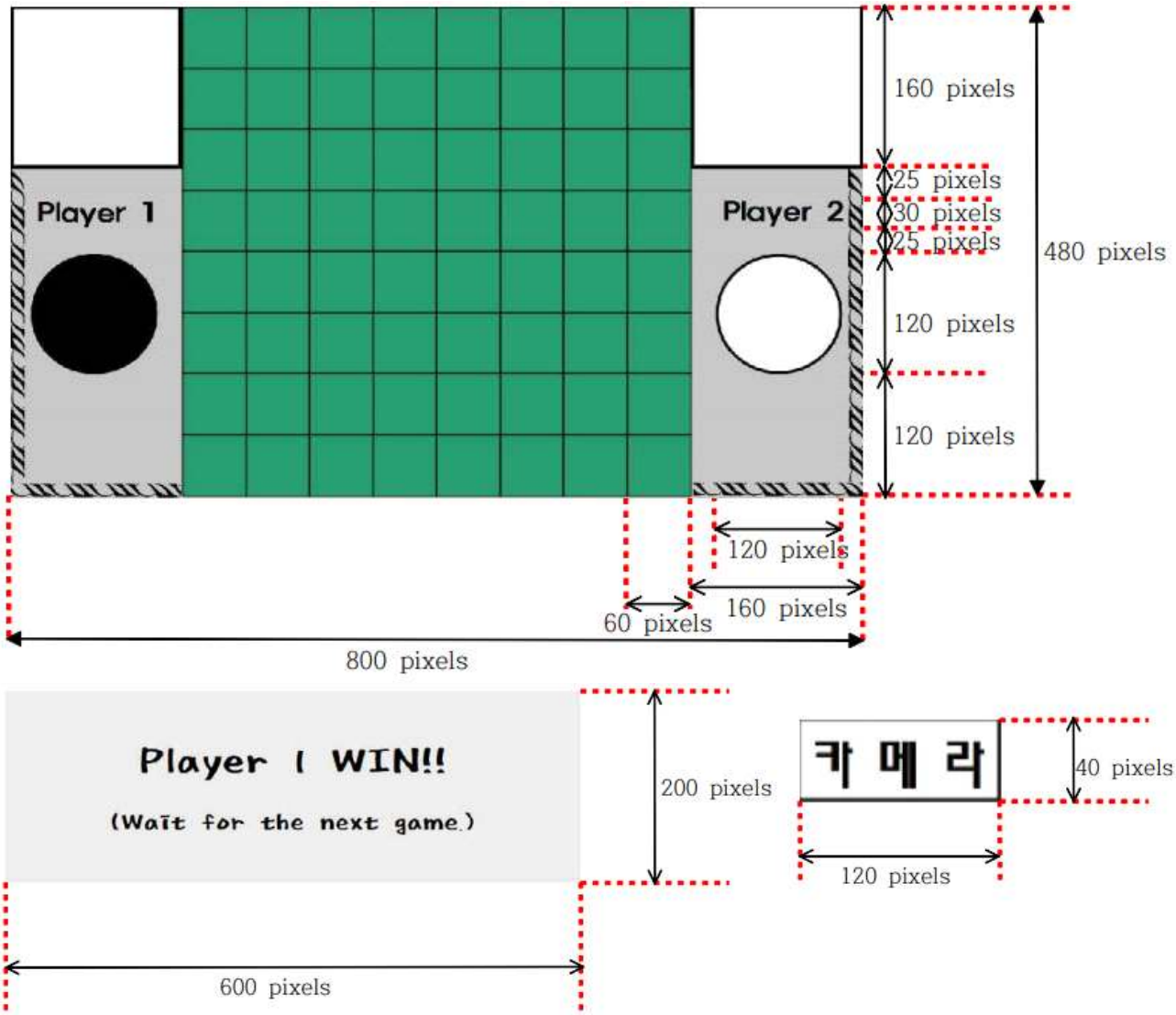
* 오셀로 구현하면서 24bits인 bitmap을 사용하는 이유?
    * 24 bits는 각 pixel은 빨, 초, 파의 값을 추려낼 수 있는 세 개의 연속된 byte열로 나타냅니다.

# 5. Implementation Phase

## A. Flow Chart



# B. 사용된 BMP 파일들




# C. Code 분석

[1] LCD 화면에 출력

```
int fb_display(unsigned short* rgb, int cols, int rows, int sx, int sy) {   // cols,rows:display 크기 sx,sy:display 위치
    int coor_x, coor_y;
    int screen_width;
    int screen_height;
    unsigned short* ptr;

    screen_width = fbvar.xres;
    screen_height = fbvar.yres;

    for (coor_y = 0; coor_y < rows; coor_y++) {
        ptr = (unsigned short*)pfbmap + (screen_width * sy + sx) + (screen_width * coor_y);
        for (coor_x = 0; coor_x < cols; coor_x++) {
            *ptr++ = rgb[coor_x + coor_y * cols];                    // pfbmap에는 직접 입력하지 않아서 잠깐 출력된다.
        }
    }
    return 0;
}
```

* rgb 파일을 기준으로 크기, 위치를 설정하여 일시적으로 LCD 화면에 출력하도록 한다.

* pfbmap은 변화가 없어서 계속 유지된다. Camera처럼 실시간으로 바뀌는 곳에 사용하기에 적합하다.

```
void Fill_Background(char* filename, int width, int height, int sx, int sy) {   // fb_display와 달리 변경하지 않은 이상 영구 출력
    FILE* bmpfd;
    char* lpImg;
    char r, g, b;
    int i, j = 0;
    bmpfd = fopen(filename, "rb");          // 파일을 읽기 모드로 열음
    if (bmpfd == NULL) {
        printf("파일 소환 실패₩n");
        exit(1);
    }

    fseek(bmpfd, 54, SEEK_SET);
    lpImg = (char*)malloc(1152000);
    fread(lpImg, sizeof(char), 1152000, bmpfd);

    for (i = 0; i < width * height; i++) {
        b = *lpImg++;
        g = *lpImg++;
        r = *lpImg++;

        frame[j] = ((r >> 3) << 11) | ((g >> 2) << 5) | (b >> 3);   // frame에 r,g,b에 해당하는 값 입력
        j++;
    }
    j = 0;
    fclose(bmpfd);

    int x, y;

    for (y = 0; y < height; y++) {
        for (x = 0; x < width; x++) {
            *(unsigned short*)(pfbmap + (x + sx) * 2 + (y + sy) * 800 * 2) = frame[x + y * width];  // pfbmap 변경
        }
    }
}
```

* fb_display와 달리 입력한 filename을 가진 파일을 기준으로 적용된다. Bmp 파일을 rgb 배열로 변경하고, 변경된 rgb 배열을 이용하여 pfbmap을 변경해준다. 즉, 추가적인 변화가 없는 이상, 계속 출력됩니다.

## [2] Enroll (플레이어 등록한다.)

```
Fill_Background("image/game.bmp", 800, 480, 0, 0);          // 첫 시작화면
Fill_Background("image/enroll.bmp", 120, 40, 20, 400);
Fill_Background("image/enroll.bmp", 120, 40, 660, 400);
dev_cam = open(CAMERA_DEVICE, O_RDWR);                      // camera on
if (dev_cam < 0) exit(1);
while (turn != 2) {
    write(dev_cam, NULL, 1);
    read(dev_cam, cis_rgb, 320 * 240 * 2);          // camera의 값 입력 받음
    RGB2cvIMG(cam_img, cis_rgb, 320, 240);
    cvRectangle(cam_img, cvPoint(80, 40), cvPoint(240, 200), CV_RGB(0, 255, 0), 2, 8, 0);   // cam_img에 사각형 추가
    cvIMG2RGB565(cam_img, cis_rgb, cam_img->width, cam_img->height);
    fb_display(cis_rgb, 320, 240, 240, 200);        // camera의 값과 사각형이 추가되서 출력
    if (GetTouch() == 1) {                          //1에 해당되는 위치를 touch하면
            for (j = 0; j < 160; j++) {
                for (i = 0; i < 160; i++) {
                    *(unsigned short*)(pfbmap + i * 2 + j * 800 * 2) = cis_rgb[(i + 80) + (j + 40) * 320];
                    p1_rgb[i + j * 160] = cis_rgb[(i + 80) + (j + 40) * 320];   // player1 image 저장
                }
            }
        turn++; // player 추가 등록, 2명이 등록되면 완료
    }
```

```
    else if (GetTouch() == 2) {
        for (j = 0; j < 160; j++) {
            for (i = 0; i < 160; i++) {
                *(unsigned short*)(pfbmap + (i + 640) * 2 + j * 800 * 2) = cis_rgb[(i + 80) + (j + 40) * 320];
                p2_rgb[i + j * 160] = cis_rgb[(i + 80) + (j + 40) * 320];   // player2 image 저장
            }
        }
        turn++;
    }
}
close(dev_cam);     // camera는 등록이외에 사용되지 않는다.
Fill_Background("image/game.bmp", 800, 480, 0, 0);
for (j = 0; j < 160; j++) {
    for (i = 0; i < 160; i++) {
        *(unsigned short*)(pfbmap + i * 2 + j * 800 * 2) = p1_rgb[i + j * 160];
        *(unsigned short*)(pfbmap + (i + 640) * 2 + j * 800 * 2) = p2_rgb[i + j * 160];
    }
}
```

* 시작할 때 배경화면으로 “image/game.bmp”를 출력하고, 플레이어 등록을 위한 button도 출력합니다. (“image/enroll.bmp”)
* Camera device를 통해 rgb 배열로 받아서, 촬영을 위한 정사각형 박스를 추가한 후에, fb_display로 계속 출력시켜줍니다.
* GetTouch()로 플레이어 1 또는 2 버튼이 입력받으면 해당 플레이어 배열에 rgb 배열을 저장하고, 상단 끝쪽에 출력합니다.
* 나머지 플레이어도 동일한 방식으로 진행하고, 총 2번 진행하면 while문이 종료됩니다.
* 이후에는 camera를 사용하지 않기 때문에 닫고, 플레이어 등록이 완료된 화면을 출력합니다.

## [3] Othello

```
typedef struct _GameBoard {
    int board[8][8];                    // 0: hint, 1: white 2: black 3:empty
    int score[3];                       // 0: empty 1: white 2: black
    int hint;                           // 둘 수 있는 자리 수
}Gameboard;

Gameboard Board = {
    {
        //board
        3, 3, 3, 3, 3, 3, 3, 3,
        3, 3, 3, 3, 3, 3, 3, 3,
        3, 3, 3, 3, 0, 3, 3, 3,
        3, 3, 3, 1, 2, 0, 3, 3,
        3, 3, 0, 2, 1, 3, 3, 3,
        3, 3, 3, 0, 3, 3, 3, 3,
        3, 3, 3, 3, 3, 3, 3, 3,
        3, 3, 3, 3, 3, 3, 3, 3
    }, 60, 2, 2, 4      // score & hint
};
```

* 8x8 보드의 상태를 나타내는 Gameboard structure.
* board[8][8]에는 각 칸의 상태를, score[0]은 빈 칸의 개수를, score[1],[2]는 각각 1번 돌(흰 돌), 2번돌(검은 돌)의 개수를, hint는 둘 수 있는 빈칸을 나타냅니다.

```
void MakeLimits(int limit[][8]) // 8방향으로 벽까지와의 거리
{
    int index, x, y;

    for (index = 0; index < 64; index++) {
        x = index % 8;
        y = index / 8;

        limit[index][0] = y;
        limit[index][1] = (y > 7 - x) ? 7 - x : y;
        limit[index][2] = 7 - x;
        limit[index][3] = (7 - y > 7 - x) ? 7 - x : 7 - y;
        limit[index][4] = 7 - y;
        limit[index][5] = (7 - y > x) ? x : 7 - y;
        limit[index][6] = x;
        limit[index][7] = (y > x) ? x : y;
    }
}
```

* 각 좌표마다 8방향으로 벽까지의 거리를 측정하기 위해 limit[64][8]을 전역변수로 사용하여 거리를 저장합니다.

```
void PutBoard(Gameboard *board, int x, int y, int turn)              // x, y: 돌 둘 위치, turn: 돌 색
{
    int ui, rx, ry, k;
    int anti = (turn ^ 0x3);                                         // 11과 XOR, 상대방
    int dx[8] = { 0,1,1,1,0,-1,-1,-1 };                              // 8방향, 12시부터 시계방향으로
    int dy[8] = { -1,-1,0,1,1,1,0,-1 };
```

```
    // part1. 돌 놓고 주변 바꿈
    if (y * 8 + x != 64) {
        board->board[y][x] = turn;                                   // 해당 위치를 해당 턴의 색으로
        board->score[0]--;                                           // 빈자리 -1
        board->score[turn]++;                                        // 둔 돌 +1

        for (ui = 0; ui < 8; ui++) {
            //r = index + dxy[ui];                                   // 위치 기준 8방향 타일
            rx = x + dx[ui];
            ry = y + dy[ui];

            for (k = 1; k < limit[y * 8 + x][ui]; ++k) {             // 벽과의 거리만큼 진행
                if (board->board[ry][rx] != anti)                    // 만약 상대방 돌이 아니라면 다음 방향으로
                    break;
                //r += dxy[ui];                                      // 방향으로 전진
                rx += dx[ui];
                ry += dy[ui];
            }
            if (board->board[ry][rx] == turn) {                  // 가다가 자기 돌 만나면
                while (--k) {                                        // 갔던만큼
                    //r -= dxy[ui];                                  // 거슬러 올라가면서
                    rx -= dx[ui];
                    ry -= dy[ui];
                    board->board[ry][rx] = turn;                     // 돌 뒤집고 내꺼+1, 상대편-1
                    board->score[turn]++;
                    board->score[anti]--;
                }

            }
        }
    }
```

* PutBoard는 돌을 뒤집거나 둘 수 있는 자리(hint) 등 전반적인 게임 룰을 다룹니다.

```
// part2. 놓을 곳 찾는 로직
board->hint = 0;

for (int y = 0; y < 8; y++) {
    for (int x = 0; x < 8; x++) {
        if (board->board[y][x] != 0 && board->board[y][x] != 3)     // 0: hint, 3: empty, 둘 다 아닐 경우 = white/black일 경우
            continue;

        board->board[y][x] = 3;                                    // 일단 빈칸으로 만들고

        for (ui = 0; ui < 8; ui++) {                               // 8방향으로 전진하면서
            rx = x + dx[ui];
            ry = y + dy[ui];

            for (k = 1; k < limit[y * 8 + x][ui]; k++) {           // 벽과의 거리만큼
                if (board->board[ry][rx] != turn)
                    break;
                rx += dx[ui];
                ry += dy[ui];
            }
            if (k != 1 && board->board[ry][rx] == anti) {
                board->board[y][x] = 0;
                board->hint++;
                break;
            }

        }
    }
}
```

```
int Othello(int first)
{
    // 시작
    unsigned int buzzer_sound;
    int x, y, turn = first,i,j;
    MakeLimits(limit);
    int W = 0, B = 0;

    while (Board.score[0] > 0) {                //Board에 둘 곳이 있는 경우

        // 보드의 시각화
        DisplayBoard(&Board, turn);
        printBoard();

        if (Board.hint != 0) {                  //플레이어의 턴에 둘 곳이 있을 경우

            while (1) {
                if (GetTouch1() != -1) {        //터치값을 받는다
                    x = y_detected + 1;
                    y = x_detected + 1;
                }
                write(dot_fd, &turn, 4);        //dotmatrix에 플레이어 표시

                if (Board.board[y - 1][x - 1] == 0) {//hint위치
                    PutBoard(&Board, x - 1, y - 1, turn);
                    break;
                }
```

```
                else if (Board.board[y - 1][x - 1] == 3) {//둘 수 없는 빈칸
                    printf("error!₩n");
                    buzzer_sound = 0xff;              //부저 소리를 0.3초 동안 실행한다
                    write(buz_fd, &buzzer_sound, 2);
                    m_delay(300);
                    buzzer_sound = 0x00;
                    write(buz_fd, &buzzer_sound, 2);
                    x = 0;
                    y = 0;
                    continue;
                }

                else {
                    continue;
                }
            }
        }
        else
            printf("₩n₩nTurn skipped₩n");
        // 턴 넘김
        if (turn == 1)
            turn++;
        else if (turn == 2)
            turn--;

        // 게임 끝났는지 확인
        if (Board.score[1] == 0 || Board.score[2] == 0)
            break;
    }


    for (i = 0; i < 8;i++) {          //백,흑의 개수를 계산한다
        for (j = 0; j < 8; j++) {
            if (Board.board[i][j] == 1) W++;
            else if (Board.board[i][j] == 2) B++;
        }
    }
    if (W > B)return 1;//백 승
    else if (W < B)return 2;//흑 승
    else return 3;//무승부
}
```

* 보드에 빈 공간이 없거나, 한 쪽 플레이어의 돌이 없는 경우 게임이 끝납니다.
* 터치패드를 이용하므로, 각 좌표의 입력은 터치값으로 계산됩니다. 이때 GetTouch1()함수를 이용하게 됩니다.
* 이미 돌이 있는곳을 제외하고 둘 수 없는곳에 두게 되는 경우에 부저가 울리게 됩니다.

```
void printBoard() {//게임의 디스크을 이미지로 출력시킨다

    int i, j;
    for (i = 0; i < 8; i++) {
        for (j = 0; j < 8; j++) {
            if (Board.board[i][j] == 1) {//흰돌
                Fill_Background("image/disk2.bmp", 60, 60, 160 + i * 60, j * 60);
            }
            else if (Board.board[i][j] == 2) {//검은돌
                Fill_Background("image/disk1.bmp", 60, 60, 160 + i * 60, j * 60);
            }
        }
    }
}
```

* printBoard함수는 오델로의 게임 데이터를, 터치패드에 이미지로 구현하는 함수입니다. Board값을 토대로 각 돌의 위치에 돌의 이미지를 덮어씌우는 방식으로 구현하였습니다.
* 이와 비슷하게 DisplayBoard 함수에서는, 콘솔창을 이용하여 각 disk의 값을 시각화하였습니다.

## [3] Touch구현 (GetTouch함수)

```
...

            case EV_ABS://터치 들어옴(절대좌표)
                switch (event_bufts[i].code) {
                case ABS_X:
                    y = event_bufts[i].value;
                    break;

                case ABS_Y:
                    x = event_bufts[i].value;
                    break;

                default:
                    break;
                }
                break;

            default:
                break;
            }

...

    if (x < X_OFFSET)
```

```
        x_detected = -1;
    else if (x < X_OFFSET + X_WIDTH)
        x_detected = 0;
    else if (x < X_OFFSET + 2 * X_WIDTH)
        x_detected = 1;
    else if (x < X_OFFSET + 3 * X_WIDTH)
        x_detected = 2;
    else if (x < X_OFFSET + 4 * X_WIDTH)
        x_detected = 3;
    else if (x < X_OFFSET + 5 * X_WIDTH)
        x_detected = 4;
    else if (x < X_OFFSET + 6 * X_WIDTH)
        x_detected = 5;
    else if (x < X_OFFSET + 7 * X_WIDTH)
        x_detected = 6;
    else if (x < X_OFFSET + 8 * X_WIDTH)
        x_detected = 7;
    else
        x_detected = -1;



    if (Y_OFFSET > y > Y_OFFSET - Y_WIDTH)//터치값이 0로 튀는 경우가 있어, 코드적으로 값을
제한했다
        y_detected = 0;
    else if (y > Y_OFFSET - 2 * Y_WIDTH)
        y_detected = 1;
    else if (y > Y_OFFSET - 3 * Y_WIDTH)
        y_detected = 2;
    else if (y > Y_OFFSET - 4 * Y_WIDTH)
        y_detected = 3;
    else if (y > Y_OFFSET - 5 * Y_WIDTH)
        y_detected = 4;
    else if (y > Y_OFFSET - 6 * Y_WIDTH)
        y_detected = 5;
    else if (y > Y_OFFSET - 7 * Y_WIDTH)
        y_detected = 6;
    else if (y > Y_OFFSET - 8 * Y_WIDTH)
        y_detected = 7;
    else
        y_detected = -1;

    return 0;
```

* GetTouch함수에서는, Touch가 있을 때 각 x, y좌표를 전체 넓이, 높이의 16000등 분의 값으로 받게 됩니다.

* 이 값을 원하는 범위 내로 이용하기 위해서 각 범위의 경계(OFFSET)과 분할 넓이(WIDTH)로 나누어 사용하였습니다.
* GetTouch()함수는 카메라 및 등록을 위한 버튼의 터치를 받기 위해서, GetTouch1() 함수는 보드의 각 칸의 터치값을 각 칸 값으로 받기 위해 사용되었습니다.

# 6. Analysis

* 이중 배열 및 structure구조를 이용한 보드게임의 구조화
* 픽셀 위치 및 터치 좌표를 이용한 좌표 입/출력
* Camera Device를 이용한 영상 출력 및 자르기
* 기타 Device를 이용하여 Condition 나타내기

# 7. Conclusion

* TouchPad를 이용한 터치좌표를 이용하여 새로운 입력을 사용하게 되었습니다.
* TouchPad 화면을 이용하기 위해 이미지파일(bmp)를 자유롭게 응용할 수 있었습니다.
* Camera 모듈을 통한 이미지와, 화면 출력을 위해 frame의 이용을 할 수 있었습니다.

# 8. References

https://m.blog.naver.com/PostView.nhn?blogId=lunchtime82&logNo=100055581202&proxyReferer=https:%2F%2Fwww.google.com%2F
https://rainbowce.tistory.com/64