Software Practice2

Programming Assignment #final:

Online Tetris

2014313303, 홍태하

1. 전체적인 흐름.
2. Single Tetris
   A. 키 입력 정상 동작, 충돌 방지
   B. 커맨드 모드
   C. 라인 제거
   D. 점수 부여 및 출력
3. 추가구현
4. 실행 화면

1. 전체적인 흐름

저는 Server-Client를 이용한 Multi Tetris는 완성하지 못하였고 Single Tetris만 완성하였습니다. 테트리스를 구현하기 위해 UI_Work(), Key_Work(), Gravity_Work() 이렇게 3가지의 thread와 printScreen이라는 함수 하나를 이용하였습니다. 또 term_control.h 헤더를 이용하였습니다.

Key_Work() 쓰레드에서는 테트리스 블록의 이동과 회전, 엔터, 스페이스 바 입력을 통한 기능 구현 등 키보드로 입력한 것들에 대해서 처리해주는 쓰레드입니다. UI_Work() 쓰레드는 키가 입력되었나 안 되었나를 체크하고 그에 따른 적절한 함수를 불러줍니다. Gravity_Work() 쓰레드는 테트리스 블록이 일정 간격으로 내려오는 것을 구현한 것입니다. printScreen은 테트리스 화면을 출력해줍니다.

2. Single Tetris

```c
enum { KEY_CHANGED_NONE = 0, KEY_CHANGED_OK, KEY_CHANGED_QUIT };

static int posX;
static int posY;
static int key_changed = KEY_CHANGED_NONE;
static int shapeNo;
static int rotationNo;
int keep = -1;
int restart_flag = 0;
int over_flag = 0;
int pause_flag=0;
int stage_num=1;
int score=0;
int end_flag=0;
char tetris[22][12] = { \
    ' ','-','-','-','-','-','-','-','-','-','-',' ',\
    '|',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ','|',\
    '|',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ','|',\
    '|',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ','|',\
    '|',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ','|',\
    '|',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ','|',\
    '|',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ','|',\
    '|',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ','|',\
    '|',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ','|',\
    '|',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ','|',\
    '|',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ','|',\
    '|',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ','|',\
    '|',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ','|',\
    '|',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ','|',\
    '|',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ','|',\
    '|',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ','|',\
    '|',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ','|',\
    '|',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ','|',\
    '|',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ','|',\
    '|',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ','|',\
    '|',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ','|',\
    ' ','-','-','-','-','-','-','-','-','-','-',' '\
};

void *UI_Work(void *arg);
void *Key_Work(void *arg);
void *Gravity_Work();

void printScreen();
```

먼저 tetris라는 배열을 잡아서 테트리스 게임을 하는 판을 전역변수로 저장해놓습니다. 그리고 테트리스 블록을 모양 7개, 회전 4가지로 해서 총 28가지의 경우로 나누어서 체크를 해주었습니다.

블록이 다른 블록이나 벽 위에 올라서 굳지 않는 경우에는 그냥 블록을 printf로 표현해주고 굳는 경우에만 tetris자체를 바꿔서 계속해서 나타나도록 해줍니다.

A. 키 입력 정상 동작, 충돌 방지

테트리스에서는 여러가지 키 입력에 대하여 처리를 해줘야 하는데 먼저 기본적으로 좌우, 아래로 테트리스 블록이 움직여야 합니다.

먼저 'w'키를 눌렀을 때는 블록이 회전하여야 합니다. 블록이 회전할 공간이 있나없나 체크한

후에 공간이 있다면 블록을 회전시킵니다.

```
if(!((tetris[posY+1][posX+2] == 'x' || tetris[posY+1][posX+2] == '|') ||
            (tetris[posY+1][posX+3] == 'x' || tetris[posY+1][posX+3] == '|') ||
            (tetris[posY+1][posX+4] == 'x' || tetris[posY+1][posX+4] == '|')) ){
    rotationNo++;
}
```

모양 0, 회전 0인 경우

'a'키를 눌렀을 때에는 블록이 왼쪽으로 이동해야 합니다. 이 경우 블록의 왼쪽에 벽이나 다른 블록이 있는지 체크한 후 이동할지 말지 결정합니다.

```
if(i == posY+1 && j == posX+1){
    if(tetris[i][j-1] == 'x' || tetris[i][j-1] == '|'){
        end_flag = 1;
    }
}else if(i == posY+2 && j == posX+1){
    if(tetris[i][j-1] == 'x' || tetris[i][j-1] == '|'){
        end_flag = 1;
    }
}else if(i == posY+3 && j == posX+1){
    if(tetris[i][j-1] == 'x' || tetris[i][j-1] == '|'){
        end_flag = 1;
    }
}else if(i == posY+4 && j == posX+1){
    if(tetris[i][j-1] == 'x' || tetris[i][j-1] == '|'){
        end_flag = 1;
    }
}
```

모양 0, 회전 0인 경우

's'키를 누른 경우 블록이 한 칸 아래로 이동해야 합니다. posY값을 1씩 늘려주면서 이동시킵니다. 아래쪽에 블록이나 벽이 있으면 멈추고 굳는데 그것은 printScreen()함수에서 체크합니다.

```
setColor(0);
if(i == posY+1 && j == posX+1){
    printf("x");
}else if(i == posY+2 && j == posX+1){
    printf("x");
}else if(i == posY+3 && j == posX+1){
    printf("x");
}else if(i == posY+4 && j == posX+1){
    printf("x");
}else if(i == posY+5 && j == posX+1){
    setColor(8);
    if(tetris[i][j] == '-' || tetris[i][j] == 'x'){
        tetris[posY+1][posX+1] = 'x';
        tetris[posY+2][posX+1] = 'x';
        tetris[posY+3][posX+1] = 'x';
        tetris[posY+4][posX+1] = 'x';

        end_flag=1;
    }
    printf("%c",tetris[i][j]);

}else{
    setColor(8);
    printf("%c", tetris[i][j]);
}
```

모양 0, 회전 0인 경우

이 경우 블록이 세로로 긴 막대모양이다. 이 블록의 바로 아래 i == posY+5, j == posX+1인 경우에 벽이나 블록이 있으면 tetris를 변형시켜 이 블록 모양이 남도록 해주고 아니면 임시적으로 printf로 처리해준다.

'd' 키를 눌렀을 때에는 블록이 오른쪽으로 이동해야 합니다. 이 경우 블록의 오른쪽에 벽이나 다른 블록이 있는지 체크한 후 이동할지 말지 결정합니다.

```
if(i == posY+1 && j == posX+1){
    if(tetris[i][j+1] == 'x' || tetris[i][j+1] == '|'){
        end_flag = 1;
    }
}else if(i == posY+2 && j == posX+1){
    if(tetris[i][j+1] == 'x' || tetris[i][j+1] == '|'){
        end_flag = 1;
    }
}else if(i == posY+3 && j == posX+1){
    if(tetris[i][j+1] == 'x' || tetris[i][j+1] == '|'){
        end_flag = 1;
    }
}else if(i == posY+4 && j == posX+1){
    if(tetris[i][j+1] == 'x' || tetris[i][j+1] == '|'){
        end_flag = 1;
    }
}
```

모양 0, 회전 0인 세로로 긴 막대 모양.

이 막대 모양의 한 칸 오른쪽이 벽이나 블록이 아닌지 체크합니다.

'₩n'엔터키를 눌렀을 경우에는 (채점 편의를 위해)모양을 순서대로 바꿔줍니다.

Space bar를 눌렀을 경우에는 그 위치에서 가장 위에 있는 블록이나 벽 위로 바로 이동합니다.

```
if(i >= posY+1){
    if(shapeNo == 0){
        if(j == posX+1){
            if(tetris[i][j] == 'x' || tetris[i][j] == '-'){
                posY = i-5;
                end_flag = 1;
                break;
            }
        }
```

세로로 긴 막대 모양일 경우 이 막대보다 밑에 있는 것들 중에서 벽이나 블록을 찾고 그 위에 위치하도록 posY값을 정해줍니다.

B. 커맨드 모드

esc키를 누르면 커맨드 모드로 변하게 됩니다. Single Tetris의 경우 커맨드 모드는 pause의 역할을 하게 됩니다. Esc를 누른 순간 하던 게임을 일시정지하고 게임을 다시 할지 게임을 그만할지 결정할 수 있게 됩니다.

```
if(key == 27){
    pause_flag = 1;
}else if(pause_flag == 1){
    if( key == 'q' ){
        key_changed = KEY_CHANGED_QUIT;
        set_disp_mode(1);
        pthread_exit(NULL);
    }else if(key == ' '){
        pause_flag = 0;
    }
```

```
if(pause_flag == 1 || over_flag == 1){
    if(key_changed == KEY_CHANGED_QUIT){
        pthread_exit(0);
    }
```

```
if(key_changed == KEY_CHANGED_QUIT){
    pthread_exit(0);
}
```

먼저 esc(아스키코드 값 27)을 누르면 pause인 것을 알리는 pause_flag를 1로 바꿔줍니다. 그리고 pause_flag가 1일 때(커맨드 모드일 때) q나 space bar를 누를 때까지 기다리고 q를 누르게 되면 3개의 쓰레드를 전부 종료시켜야 합니다. 위 그림 위에서부터 Key_Work, Gravity_Work, UI_Work쓰레드입니다. 또 스페이스 바를 누르면 pause_flag를 다시 0으로 바꿔줘서 pause를 풀고 다시 게임을 하게 됩니다.

C. 라인 제거

라인은 한번에 최대 4개 최소 1개 제거할 수 있습니다.

```
for(i=21;i>0;i--){
    for(j=0;j<12;j++){
        if(tetris[i][j] == 'x'){
            count++;
        }
        if(count == 10){
            index_len++;
            for(k=0;k<4;k++){
                if(index[k] == 0){
                    index[k] = i;
                    break;
                }
            }
            break;
        }
    }
    count = 0;
}
```

라인을 최대 4개까지 제거할 수 있으므로 index[4]라는 4칸짜리 배열을 만들고 이 배열에 제거할 라인의 줄 index를 저장합니다.

제거할 라인은 한 줄씩 보면서 카운트를 세서 블록이 10개가 있으면 꽉 찬 것이므로 제거합니다.

Index_len은 점수 부여를 위해 제거할 라인이 몇 개인지 셉니다.

```
for(k=3;k>=0;k--){
    if(index[k] != 0){
        for(i=index[k]-1;i>0;i--){
            for(j=0;j<12;j++){
                tetris[i+1][j] = tetris[i][j];
            }
        }
    }
}
```

index안의 값이 0이 아닌 경우에 그 인덱스에 해당하는 줄을 없애고 위에서 한 칸씩 당겨옵니다.

D. 점수 부여 및 출력

점수 부여는 한 번에 1줄 제거했을 경우 10점, 2줄 제거했을 경우 40점, 3줄 제거했을 경우 120점, 4줄 제거 했을 경우 320점을 부여합니다. score라는 변수를 전역으로 선언하여 라인을 제거할 때마다 더해줍니다.

```
if(index_len == 1){
    score = score + 10;
}else if(index_len ==2){
    score = score + 40;
}else if(index_len == 3){
    score = score + 120;
}else if(index_len == 4){
    score = score + 320;
}
```

출력은 테트리스 게임을 하는 테두리 위에 stage와 score가 표시되어 있고 테트리스는 22*12짜리

배열 안에서 진행합니다. 테트리스 게임 테두리 오른쪽에는 keep해놓은 블록이 표시되어 있습니다.

3. 추가구현

A. 블록 색

먼저 7가지 블록의 모양에 따라서 각각 7가지 색을 넣었습니다. setColor()를 이용하여 블록을 프린트할 때 색을 다르게 하여 보기 편하게 하였습니다.

B. Restart 기능

게임 오버가 되었을 때 원래는 q를 통해 나가는 기능밖에 없지만 엔터 키를 누르면 restart할 수 있는 기능을 추가했습니다.

```
for(j=0;j<12;j++){
    if(tetris[1][j] == 'x'){
        clearScreen();
        setColor(1);
        printf("Stage : %d\n",stage_num);
        printf("Score : %d\n",score);
        printf(" "); for(i=0;i<10;i++){printf("-");}printf(" \n");
        printf("|"); for(i=0;i<10;i++){printf(" ");}printf("|\n");
        printf("|"); for(i=0;i<10;i++){printf(" ");}printf("|\n");
        printf("|"); for(i=0;i<10;i++){printf(" ");}printf("|\n");
        printf("|"); for(i=0;i<10;i++){printf(" ");}printf("|\n");
        printf("|"); for(i=0;i<10;i++){printf(" ");}printf("|\n");
        printf("|"); for(i=0;i<3;i++){printf(" ");}printf("Game");for(i=0;i<3;i++){printf(" ");}printf("|\n");
        printf("|"); for(i=0;i<3;i++){printf(" ");}printf("Over");for(i=0;i<3;i++){printf(" ");}printf("|\n");
        printf("|"); for(i=0;i<10;i++){printf(" ");}printf("|\n");
        printf("|"); for(i=0;i<1;i++){printf(" ");}printf("Restart");for(i=0;i<2;i++){printf(" ");}printf("|\n");
        printf("|"); for(i=0;i<2;i++){printf(" ");}printf(":Enter");for(i=0;i<2;i++){printf(" ");}printf("|\n");
        printf("|"); for(i=0;i<10;i++){printf(" ");}printf("|\n");
        printf("|"); for(i=0;i<3;i++){printf(" ");}printf("Quit");for(i=0;i<3;i++){printf(" ");}printf("|\n");
        printf("|"); for(i=0;i<1;i++){printf(" ");}printf(":press q");for(i=0;i<1;i++){printf(" ");}printf("|\n");
        printf("|"); for(i=0;i<10;i++){printf(" ");}printf("|\n");
        printf("|"); for(i=0;i<10;i++){printf(" ");}printf("|\n");
        printf("|"); for(i=0;i<10;i++){printf(" ");}printf("|\n");
        printf("|"); for(i=0;i<10;i++){printf(" ");}printf("|\n");
        printf("|"); for(i=0;i<10;i++){printf(" ");}printf("|\n");
        printf("|"); for(i=0;i<10;i++){printf(" ");}printf("|\n");
        printf("|"); for(i=0;i<10;i++){printf(" ");}printf("|\n");
        printf(" "); for(i=0;i<10;i++){printf("-");}printf(" \n");
        over_flag=1;
        setColor(8);
    }
}
```

tetris배열의 맨 위에 블록이 있을 경우 게임오버가 됩니다. 게임오버가 되면 게임오버가 된 화면을 띄우고 over_flag를 1로 바꿔줍니다.

```
}else if(over_flag == 1){
    if( key == 'q' ){
        key_changed = KEY_CHANGED_QUIT;
        set_disp_mode(1);
        pthread_exit(NULL);
    }else if(key == '\n'){
        key_changed = KEY_CHANGED_OK;
        restart_flag = 1;
        over_flag = 0;
    }
```

Key_Work() 쓰레드에서 over_flag가 1인 경우 q를 통해 나가거나 엔터를 통해 restart할 수 있는데 엔터를 누르면 restart_flag를 1로 바꿔주고 over_flag는 다시 0으로 초기화 해줍니다.

```
clearScreen();

if(restart_flag == 1){
    tetris[0][0]=' ';
    tetris[0][11]=' ';
    tetris[21][0]=' ';
    tetris[21][11]=' ';
    for(j=1;j<11;j++){
        tetris[0][j]='-';
        tetris[21][j]='-';
    }

    for(i=1;i<21;i++){
        tetris[i][0]='|';
        tetris[i][11]='|';
    }
    for(i=1;i<21;i++){
        for(j=1;j<11;j++){
            tetris[i][j]=' ';
        }
    }
    posY = 0;
    posX = 0;
    shapeNo = rand() % 7;
    restart_flag = 0;
}
```

Restart_flag가 1이 되면 tetris를 다시 맨 처음 상태로 바꿔주고 posY와 posX를 맨 위로 올려준 뒤 모양도 랜덤으로 다시 뽑고 restart_flag는 0으로 초기화 해줍니다. 그 후에는 다시 정상동작하여 게임을 진행할 수 있습니다.

C. Stage

Stage를 5단계로 나누어서 stage가 올라갈수록 내려오는 블록의 속도를 빠르게 하였습니다.

```
}else if(score <=200){
    sleep(1);
    posY++;
    stage_num = 1;
    printScreen();
}else if(score <=500){
    usleep(1000*1000*0.8);
    posY++;
    stage_num = 2;
    printScreen();
}else if(score <=1000){
    usleep(1000*1000*0.6);
    posY++;
    stage_num = 3;
    printScreen();
}else if(score <=2000){
    usleep(1000*1000*0.4);
    posY++;
    stage_num = 4;
    printScreen();
}else{
    usleep(1000*1000*0.2);
    posY++;
    stage_num = 5;
    printScreen();
}
```

Gravity_Work() 쓰레드에서 스코어에 따라 속도를 1초부터 0.2초씩 빠르게 떨어지도록 설정하였습니다.

D. 블록 keep

블록 keep은 기존의 테트리스에 있는 기능으로 자신이 keep해두고 싶은 블록이 있으면 keep해 놨다가 나중에 그 블록으로 바꿔서 사용할 수 있는 기능입니다.

```
}else if(key == 'f'){
    if(keep == -1){
        keep = shapeNo;
        shapeNo = rand()%7;
        posX=5;
        posY=-1;
    }else{
        shapeNo = keep;
        posX=5;
        posY=-1;
        keep = -1;
    }
```

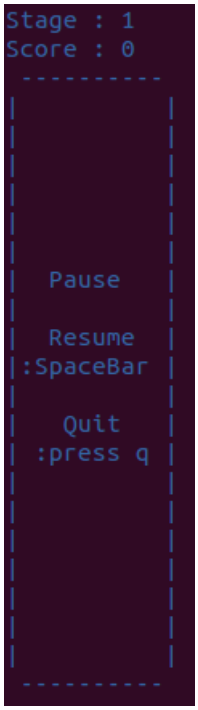
저는 방향키와 가까운 위치에 있는 f키를 누르면 동작하도록 했는데 f를 누르면 keep이라는 변수에 shapeNo를 저장합니다. 저장하지 않을 때는 -1을 값으로 가지고 있습니다. Keep이 -1일 경우에 shapeNo를 저장하고 posX,posY를 다시 위로 올려줍니다. 블록이 keep되어 있는 상태에서 다시 f를 누르면 keep하고 있던 블록으로 바뀌고 keep은 -1로 다시 초기화 됩니다.

또 테트리스 창 옆에 keep되어 있는 블록을 표시하는 칸을 만들어 표시합니다.

4. 실행 화면

스테이지와 스코어, 테트리스 실행창과 keep블록이 표시되어 있습니다.

라인을 제거하면 스코어가 오르고 스코어에 따라서 스테이지가 오릅니다. 또 keep한 블록을 볼 수 있습니다.



커맨드 상태입니다.

게임이 일시정지 되고 게임을 계속할지, 그만할지 정할 수 있습니다.

게임오버 된 모습입니다. 블록의 위쪽 벽까지 닿으면 game over 가 됩니다. 다시 게임을 할 지 나갈 지 정할 수 있습니다.