전공: 국제한국학 학년: 4 학번: 20181202 이름 : 김수미

1. 실습 시간에 작성한 프로그램의 함수들이 예비보고서에서 작성한 각 구현 함수들의 pseudo code와 어떻게 달라졌는지 설명하고, 각 함수에 대한 시간 및 공간 복잡도를 보이시오 (각 함수의 시간 및 공간복잡도를 구할 때 어떤 변수에 의존하는지를 판단해야 한다).

① int CheckToMove

4x4 '블록 필드'를 첫번째 행부터 차례대로 훑으며 '블록'이 있는 칸에 대해 해당 칸이 필드에 표시될 수 있는지를 검사한다
- 블록 == 1 인 필드가 필드 == 1인지 Check
- 블록의 현재 Y좌표 >= 필드의 HEIGHT 인지 Check
- 블록의 X 좌표 < 0 인지 또는 >= WIDTH 인지 Check

```
int CheckToMove(char f[HEIGHT][WIDTH],int currentBlock,int blockRotate,int blockY,int blockX){
    int i, j; // 블록은 4x4 행렬이므로 이중 loop 사용
    for (i = 0; i < BLOCK_HEIGHT; i++){
        for (j = 0; j < BLOCK_WIDTH; j++){
            if (block[currentBlock][blockRotate][i][j] == 1){
                // 1) 블록을 놓으려고 하는 필드에 이미 블록이 있는지
                if(f[blockY + i][blockX + j] == 1) return 0;
                // 2) 블록 요소가 실제 필드상의 y 좌표가 Height 보다 크거나 같은지
                else if ((blockY + i) >= HEIGHT) return 0;
                // 3) 블록 요소가 실제 필드상의 x 좌표가 0보다 작은지
                else if ((blockX + j) < 0) return 0;
                // 4) 블록 요소가 실제 필드상의 x 좌표가 WIDTH보다 크거나 같은지
                else if ((blockX + j) >= WIDTH) return 0;
            }
        }
    }
    return 1;
}
```

(위에서부터 차례대로 pseudo code, 프로그램 구현) 프로그램과 수도코드가 구조적으로 큰 차이는 없기 때문에, 수도코드를 어떻게 구현했는지를 위주로 설명을 작성하였다.

- 수도코드에서 '4x4 블록 필드를 첫번째 행부터 차례대로 훑는' 것은 이중 루프로 구현했으며, 블록이 이미 있는 칸인지의 여부는 if (block[currentBlock][blockRotate][i][j] == 1) 의 조건문을 이용하여 조사했다.
- 블록필드에서 1의 값이 있는 위치가 [i][j] 일 때 필드 내에서 [blockY + i][blockX + j]의 위치를 가지므로 이를 활용해 블록의 현재 좌표가 필드 내부에 위치하는지를 check 한다. 필드 내부에 위치하며 해당 자리에 다른 블록이 없는 경우 해당 함수는 1을 반환하고, 필드를 벗어나거나 해당 자리에 이미 다른 블록이 있는 경우 0을 반환한다. 즉 블록 필드 내에 있는 블록 중에서 하나라도 잘못된 자리에 위치한 것이 있다면 해당 함수의 return 값은 0이 되고, 블록은 CheckToMove 함수의 매개변수로 받은 위치로 이동할 수 없음을 의미한다.
- 시간복잡도 : 변수 BLOCK_HEIGHT와 BLOCK_WIDTH에 의존한다. 이중 루프 내부의 계산은 모두 상수 값의 시간복잡도를 가지므로, 최종 시간복잡도는 루프가 몇 번을 도는지에 달려있으며 해당 횟수는 (BLOCK_HEIGHT× BLOCK_WIDTH) 이므로 O(BLOCK_HEIGHT×BLOCK_WIDTH)의 시간복잡도를 가진다.
- 공간복잡도 : 현재 두개의 이차원 배열(블록 필드, 전체 필드)을 비교하고 있으므로 두가지 배열이 생성되었다는 가정 하에 CheckToMove 함수가 호출됨을 알 수 있다. 따라서 두 배열 중 더 큰 전체 필드의 크기에 공간복잡도가 upper bound 되며 이는 O(HEIGHT×WIDTH) 이고 변수 HEIGHT 와 WIDTH에 의존한다.

전공: 국제한국학 학년: 4 학번: 20181202 이름 : 김수미

② void DrawChange

변경된 블록 회전수, Y좌표, X좌표를 매개변수로 받으므로
- 먼저 변경되기 전의 블록 모양을 찾아 그것을 필드 위에서 지운다
- 변경된 후의 블록 모양을 필드 위에 그린다

```
void DrawChange(char f[HEIGHT][WIDTH], int command, int currentBlock, int blockRotate, int blockY, int blockX){
    // 1) 이전 블록 정보를 찾는다. ProcessCommand의 switch문을 참조할 것
    int tempR = blockRotate, tempY = blockY, tempX = blockX;
    switch(command){
    case KEY_UP:
        tempR = (tempR+3) % 4; break;
    case KEY_DOWN:
        tempY--; break;
    case KEY_RIGHT:
        tempX--; break;
    case KEY_LEFT:
        tempX++; break;
    }
    // 2) 이전 블록 정보를 지운다. DrawBlock함수 참조할 것
    for (int i = 0; i < BLOCK_HEIGHT; i++){
        for (int j = 0; j < BLOCK_WIDTH; j++){
            if (block[currentBlock][tempR][i][j] == 1 && tempY+i >= 0){
                move(i+tempY+1, j+tempX+1); // (DrawBlock에서 가져온 것)
                printw(".");
            }
        }
    }
    // 3) 새로운 블록 정보를 그린다.
    DrawBlock(blockY, blockX, currentBlock, blockRotate, ' ');
}
```

- DrawChange 함수는 변경되기 전 블록의 위치를 찾아 해당 위치에 기록된 블록 정보를 필드에서 지우고, 변경된 후의 블록 위치를 필드 위에 기록하는 역할을 하는 함수이다.

- 변경되기 전 블록 모양과 위치는 command 매개변수를 통해 유추한다. 예를 들어 오른쪽으로 한 칸 움직이는 command를 받은 경우 기존 블록의 모양은 지금과 같고, 위치만 한 칸 왼쪽에 있었음을 유추할 수 있다.

- 변경되기 전 블록 모양과 위치를 파악했으면 CheckToMove함수에서와 동일한 방법으로 해당 블록 필드 위 블록의 위치를 필드에서 찾아, 그 위치에는 블록이 있었음을 표시하는 1이 있을테니 이를 모두 0으로 바꿔 필드 위 기존 블럭의 정보를 삭제한다.

- 이제 새로운 블럭을 필드에 그려넣을 차례이다. DrawChange 함수의 매개변수가 새로운 블럭에 대한 정보를 이미 가지고 있으므로 전달받은 매개변수를 그대로 사용하면 된다. 이 때 DrawBlock 함수를 호출한다.

- 시간복잡도 : 변경되기 전 블록의 위치를 찾는 부분은 상수시간을 가지지만, 블록 정보를 필드에서 지우는 부분과 새로운 블록을 필드에 기록하는 부분이 각각 4x4 블록 필드를 모두 훑는 과정을 거치므로 전체적인 시간복잡도는 변수 BLOCK_HEIGHT와 BLOCK_WIDTH에 의존하며 O(BLOCK_HEIGHT×BLOCK_WIDTH)에 bound 된다.

- 공간복잡도 : 해당 함수에서도 전체 필드를 access 하는 부분이 있기 때문에 전체적인 공간복잡도는 변수 HEIGHT 와 WIDTH에 의존하며 O(HEIGHT×WIDTH)에 bound 된다.

전공: 국제한국학 학년: 4 학번: 20181202 이름 : 김수미

③ int BlockDown

1초마다 수행되는 함수, 매 초 블록이 아래로 떨어질 수 있는지를 check 하고 이를 수행
- 아래로 떨어질 수 없는 경우 : 필드에 블록을 쌓거나 게임오버
- 게임오버가 아닌 경우 : 가득 차는 행이 있는지 검사 후 있다면 지우고 점수 부여
- 현재 블록은 필드에 정보를 표시 후 위치 초기화
- 다음에 떨어질 블록으로 바꿔주기

+) 줄이 지워질 때(100점), 블록이 필드에 쌓일 때(10점) 마다 점수 부여 후 화면에 프린트

```
void BlockDown(int sig){
    // 강의자료 p26-27의 flowchart 참고
    // 매개변수 int sig 는 신호를 받기 위함 이므로 함수 내부에서 사용하지 않아도 됨
    // 왼쪽부터 순서대로 시간복잡도, 공간복잡도
    timed_out = 0;

    if (CheckToMove(field, nextBlock[0], blockRotate, blockY+1, blockX)==1){ // O(BLOCK_HEIGHT×BLOCK_WIDTH), O(HEIGHT×WIDTH)
        DrawChange(field, KEY_DOWN, nextBlock[0], blockRotate, ++blockY, blockX); // O(BLOCK_HEIGHT×BLOCK_WIDTH), O(HEIGHT×WIDTH)
        return;
    }
    else {
        if (blockY == -1) gameOver = 1;
        AddBlockToField(field,nextBlock[0],blockRotate,blockY,blockX); // O(BLOCK_HEIGHT×BLOCK_WIDTH), O(HEIGHT×WIDTH)
        score += 10; // 블럭을 필드에 쌓을때마다 +10점
    }

    // 한 줄을 가득 채워 없애는 경우 +100점
    score += DeleteLine(field); // O(HEIGHT×WIDTH), O(HEIGHT×WIDTH)
    PrintScore(score); // O(1)

    // 현재 블럭을 다음 블럭으로 바꾸고, 다음 블럭 업데이트
    nextBlock[0] = nextBlock[1];
    nextBlock[1]=rand()%7;
    DrawNextBlock(nextBlock); // O(BLOCK_HEIGHT×BLOCK_WIDTH), O(BLOCK_HEIGHT×BLOCK_WIDTH)

    // 현재 블록의 위치를 초기화
    blockX = WIDTH/2 -2;
    blockY = -1;
    blockRotate = 0;
    DrawField(); // O(HEIGHT×WIDTH), O(HEIGHT×WIDTH)

}
```

- BlockDown 함수는 play 함수에서 act.sa_handler = BlockDown 식으로 호출되는 함수로, 1초마다 한 번씩 호출되어 수행되는 함수이다.
- 테트리스 블록이 아래로 한 칸 떨어질 수 있는지를 Check. 떨어질 수 있다면 블록을 한 칸 내리고 함수를 종료한다. 이 때 앞에서 작성한 CheckToMove 함수와 DrawChange 함수가 사용된다
- 만약 테트리스 블록이 아래로 한 칸 움직일 수 없다면, 1) 게임오버인지를 Check = blockY 값이 -1 인지 Check. 맞으면 gameOver 변수 값을 1로 set 되고 play 함수 내부에서 게임이 종료된다. 2) 게임오버가 아니라면 이미 필드에 자리하고 있는 다른 테트리스 블록 위에 현재 블록이 놓인 것이므로, 필드에 블록의 현재 위치를 기록해 필드 위에 블록을 쌓는다. 이 때 AddBlockToField 함수가 사용된다
- 블록의 위치 업데이트가 완료되었으면 1) 필드 배열 전체를 훑어 가득 찬 행이 생겼는지를 Check. 생겼다면 해당 행을 지우고 점수를 부여한 뒤 상단의 행들을 한 줄씩 내린다. 2) 다음에 나올 블럭을 준비시킨다 : next block을 current block 으로 가져오고, next block 정보는 다시 랜덤으로 업데이트 3) 필드에 블록 정보가 기록되었으므로 현재 블록의 위치는 초기화해도 된다. 즉 blockX, blockY, blockRotate 변수를 초기화시키고 지금까지 변경된 값들을 기반으로 필드를 업데이트 한다 = DrawField 함수 호출. 이는 0에서 1로 바뀐 필드의 위치에 '■'를 그려넣는 역할을 한다.
- 시간복잡도 : O(HEIGHT×WIDTH) / 변수 HIEGHT, WIDTH 에 의존
- 공간복잡도 : O(HEIGHT×WIDTH) / 변수 HIEGHT, WIDTH 에 의존

BlockDown이 호출하는 함수 중 시공간 복잡도가 가장 큰 함수의 복잡도에 bound 된다. (주석참고)

전공: 국제한국학 학년: 4 학번: 20181202 이름 : 김수미

④ void AddBlockToField

현재 블럭이 자리잡은 위치의 필드값을 1 로 바꾼다. 블럭의 위치가 초기화되어도 필드에는 그 정보가 남아있도록

```
void AddBlockToField(char f[HEIGHT][WIDTH],int currentBlock,int blockRotate,int blockY,int blockX){
    // Block이 추가된 영역의 필드값을 바꾼다.
    int i, j;
    for (i = 0; i < BLOCK_HEIGHT; i++){
        for (j = 0; j < BLOCK_WIDTH; j++){
            if (block[currentBlock][blockRotate][i][j] == 1)
                f[blockY + i][blockX + j] = 1;
        }
    }
}
```

앞서 살펴본 BlockDown 함수 내부에서 호출되는 함수이다. BlockDown 함수에서 블록이 아래로 내려가고자 하는 경로에 이미 블록이 있는 경우, 이동을 중지시키고 안착한 위치에 블록을 고정시켜주는(필드에 기록해주는) 부분에서 필드 기록을 담당한다.

4x4 블록 필드를 훑으면서 블록이 있는 부분 [i][j]는 필드에서 [blockY + i][blockX + j] 이므로 해당 위치에 블록이 있으면 이에 해당하는 필드 위치의 값을 0에서 1로 바꿔준다. 이렇게 1로 바뀐 부분들은 BlockDown 함수에서 DrawField 함수가 호출될 때 '■'가 호출된다.

- 시간복잡도 : O(BLOCK_HEIGHT×BLOCK_WIDTH) / 변수 HIEGHT, WIDTH 에 의존

〉〉 블록 필드를 이중 루프를 통해 훑기 때문에 전체적인 시간복잡도는 위와 같다.

- 공간복잡도 : O(HEIGHT×WIDTH) / 변수 HIEGHT, WIDTH 에 의존

〉〉 블록 필드와 전체 필드가 변수로써 호출되므로 두 배열이 이미 저장공간에 할당되어 있다는 것을 전제로 한다. 따라서 크기가 더 큰 전체 필드의 크기인 HEIGHT×WIDTH 에 공간복잡도는 bound 된다.

전공: 국제한국학 학년: 4 학번: 20181202 이름 : 김수미

⑤ int DeleteLine

10x22 배열의 필드를 한 줄씩 훑으며 전체 행이 1인 것이 있다면 해당 행을 삭제 + 점수 부여
= 상단의 필드를 한 칸씩 아래로 내려 덮기

```
int DeleteLine(char f[HEIGHT][WIDTH]){
    // 1) 필드에 꽉 찬 구간이 있는지 탐색
    int flag, score = 0;
    for (int i = 0; i < HEIGHT; i++){ // HEIGHT = 22
        flag = 0;
        for (int j = 0; j < WIDTH; j++){ // WIDTH = 10
            if(f[i][j]==0) { // 현재 탐색중인 행에 빈 자리가 있음
                flag = 1;
                break;
            }
        }
        // 2) 꽉 찬 구간이 있으면(i번째 행) 해당 구간을 지운다(해당 구간으로 필드값을 한칸씩 내린다)
        if(flag == 0) {
            for(int k=i; k>=0; k--){
                for(int l=0; l<WIDTH; l++) {
                    if(k==0) f[k][l] = 0;
                    else f[k][l] = f[k-1][l];
                }
            }
            // 3) 점수 갱신 후 출력
            score += 100;
        }
    }
    return score;
}
```

BlockDown 함수에서 호출되는 함수로, 가득 찬 행이 있는지를 체크하고 있다면 해당 라인을 지우는,
즉 해당 라인 상단의 행들을 모두 한 줄씩 내리는 역할을 수행한다. 지우는 라인 상단의 행들을 한 줄씩 내리면 필드의 가장 첫번째 줄은 초기화가 안되기 때문에 따로 0 으로 값을 채워주었다.

- 시간복잡도 : O(HEIGHT×WIDTH) / 변수 HIEGHT, WIDTH 에 의존

〉〉 이중 루프를 돌면서 이차원 배열로 구성된 필드의 모든 요소들을 훑으면서 가득 찬 행이 있는지를 체크하므로, 시간복잡도는 위와 같으며 필드를 구성하는 모든 요소의 개수, 즉 필드 세로(HEIGHT), 가로(WIDTH) 길이의 곱에 시간복잡도가 bound 된다.

- 공간복잡도 : O(HEIGHT×WIDTH) / 변수 HIEGHT, WIDTH 에 의존

〉〉 필드의 요소들을 모두 확인하기 때문에 필드의 크기 HEIGHT×WIDTH 에 공간복잡도가 bound 된다.

실험 PRJ-1 1주차 기본 테트리스 프로그램 결과보고서

전공: 국제한국학 학년: 4 학번: 20181202 이름 : 김수미

2. 테트리스 프로젝트 1주차 숙제 문제를 해결하기 위한 Pseudo code를 기술하고, 작성한 Pseudo code의 시간 및 공간 복잡도를 보이시오.

1) 그림자 기능 구현

① **DrawShadow 함수** : 현재 블록의 위치를 매개변수로 받고, 해당 위치에서 아래로 내릴 수 있는 만큼 블록을 내린다. 그리고 그 위치에 그림자를 그린다.

while (CheckToMove(현재 블록 위치 y좌표 +1)==1) y좌표++;

DrawBlock(y좌표, x좌표, 블럭 모양, 회전수, '/'); // 그림자는 일반 블럭과 달리 '/' 무늬로 채워진다

- 시간/공간복잡도 : O(BLOCK_HEIGHT×BLOCK_WIDTH), O(HEIGHT×WIDTH)

〉〉 CheckToMove 함수의 시공간 복잡도가 O(BLOCK_HEIGHT×BLOCK_WIDTH), O(HEIGHT×WIDTH) 이고 DrawBlock 함수의 시공간 복잡도가 O(BLOCK_HEIGHT×BLOCK_WIDTH), O(BLOCK_HEIGHT×BLOCK_WIDTH) 이므로 DrawShadow 함수의 시공간 복잡도는 O(BLOCK_HEIGHT×BLOCK_WIDTH), O(HEIGHT×WIDTH) 에 bound 된다.

② **DrawBlockWithFeatures 함수** : 단순히 DrawBlock 함수와 DrawShadow 함수를 호출하는 역할

DrawShadow( );

DrawBlock(' '); // 일반 블럭 무늬로 채운다

- 시간/공간복잡도 : O(BLOCK_HEIGHT×BLOCK_WIDTH), O(HEIGHT×WIDTH)

〉〉 DrawShadow 함수의 시공간 복잡도가 O(BLOCK_HEIGHT×BLOCK_WIDTH), O(HEIGHT×WIDTH) 이고 DrawBlock 함수의 시공간 복잡도가 O(BLOCK_HEIGHT×BLOCK_WIDTH), O(BLOCK_HEIGHT×BLOCK_WIDTH) 이므로 DrawBlockWithFeatures 함수의 시공간 복잡도는 O(BLOCK_HEIGHT×BLOCK_WIDTH), O(HEIGHT×WIDTH) 에 bound 된다.

2) 2개의 다음 블록을 미리 보는 기능 구현

① **InitTetris 함수 수정** : nextBlock[2]=rand()%7 추가

- 시간/공간복잡도 : 전체 필드를 훑으면서 필드 값을 0으로 초기화 시켜주는 작업이 포함되어 있으므로 O(HEIGHTxWIDTH), O(HEIGHTxWIDTH) 의 시간/공간복잡도를 갖는다.

② **DrawNextBlock 함수 수정** : nextBlock[2]을 필드 위에 그리는 과정 추가

for(블럭 필드를 위에서 아래로 훑는다){

두번째 다음 블럭을 그릴 위치로 커서 이동;

for(블럭 필드를 좌에서 우로 훑는다){

if(블럭 필드에 블럭이 있다 == 1) 네모(■) 하나 그리기;

else (공백 그리기); } }

전공: 국제한국학 학년: 4 학번: 20181202 이름 : 김수미

- 시간/공간복잡도 : 4X4의 블럭 필드를 훑으면서 1값이 있는 경우 이를 화면에 출력해주므로 O(BLOCK_HEIGHT×BLOCK_WIDTH), O(BLOCK_HEIGHT×BLOCK_WIDTH)의 시간/공간복잡도를 갖는다.

③ **BlockDown 함수 수정** : 아래와 같이 nextBlock이 하나 추가된 모양으로 수정해주면 된다.

```
nextBlock[0] = nextBlock[1];
nextBlock[1] = nextBlock[2];
nextBlock[2] = rand( )%7
```

- 시간/공간복잡도 : 수정 후에도 여전히 앞에서 설명한 O(HEIGHT×WIDTH), O(HEIGHT×WIDTH)의 시간/공간복잡도를 갖는다.

3) 필드와 블록의 아래 부분이 맞닿은 면적의 수에 비례하여 점수를 증가시키는 기능 구현

① **AddBlockToField 함수 수정** : return 값이 int 값이 될 수 있도록 함수의 type을 먼저 수정해준다. 그 다음 블록 필드를 차례대로 훑으면서 블럭이 있는 부분을 필드 위에 표시할 때, 해당 위치 바로 아래 부분의 필드값이 1인지를 검사하고, 1이 맞다면 touched 변수에 +1을 해 최종적으로 touched 변수 x 10 만큼의 점수를 부여하는 부분을 추가한다.

```
for(int i = 0; i<WIDTH; i++) f[HEIGHT][i] = 1; // 필드 맨 아래 테두리 부분의 필드값을 1로 채워준다
for (int i = 0; i < BLOCK_HEIGHT; i++) {
        for (int j = 0; j < BLOCK_WIDTH; j++) {
                if (블럭필드에 블럭이 있으면) {
                        필드[blockY + i][blockX + j] = 1;
                        if(위에서 1로 셋팅한 위치 바로 아래 필드값이 1일 때) touched++; } } }
점수 = touched * 10;
return 점수;
```

- 시간/공간복잡도 : 위 기능을 추하는 부분은 아무런 루프를 포함하지 않으므로 시간/공간복잡도는 기존과 동일하게 O(BLOCK_HEIGHT×BLOCK_WIDTH), O(HEIGHT×WIDTH)이 된다.