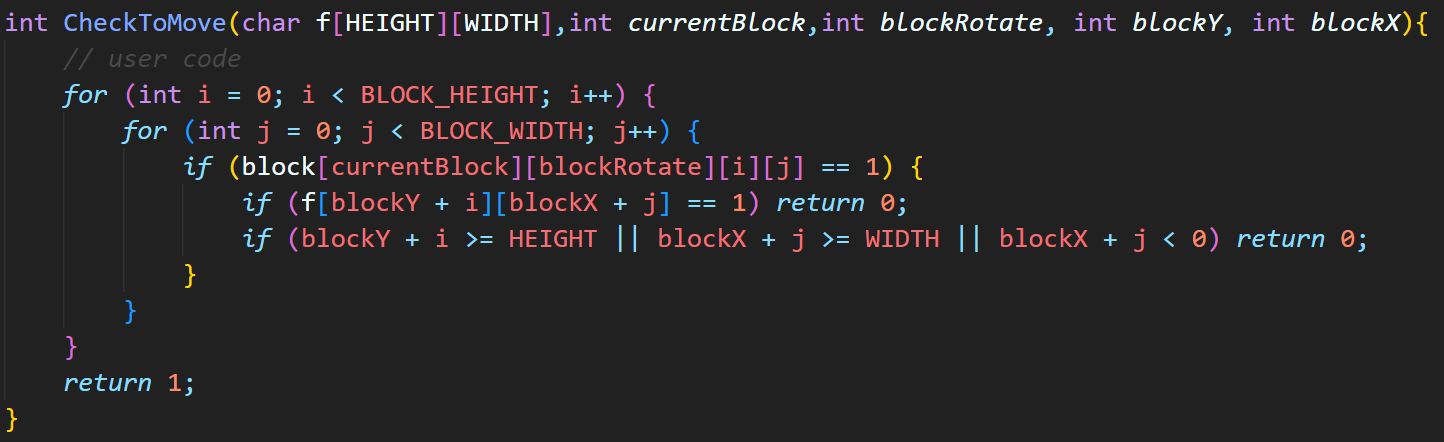
전공 : 컴퓨터공학과 학년 : 4 학번 : 20212020 이름 : 박민준

1. 실습 시간에 작성한 프로그램의 함수들이 예비보고서에서 작성한, 각 구현 함수들의 pseudo code와 어떻게 달라졌는지 설명하고, 시간 및 공간 복잡도를 보이시오.

1) CheckToMove() 함수



- Pseudo code와 코드 흐름은 비슷하지만, 0을 반환하는 경우를 크게 4가지로 구분하여 코드를 작성하였다. 해당 함수는 이중 for문을 사용하여 블록의 모든 셀을 확인한다.

- 각 셀이 1인 경우, 즉 블록이 채워진 경우, 해당 셀이 게임 필드의 경계를 넘거나 다른 블록과 충돌하는지를 확인한다. 만약 충돌이 발생하거나 필드의 경계를 넘으면 0을 반환하고, 그렇지 않으면 1을 반환한다.

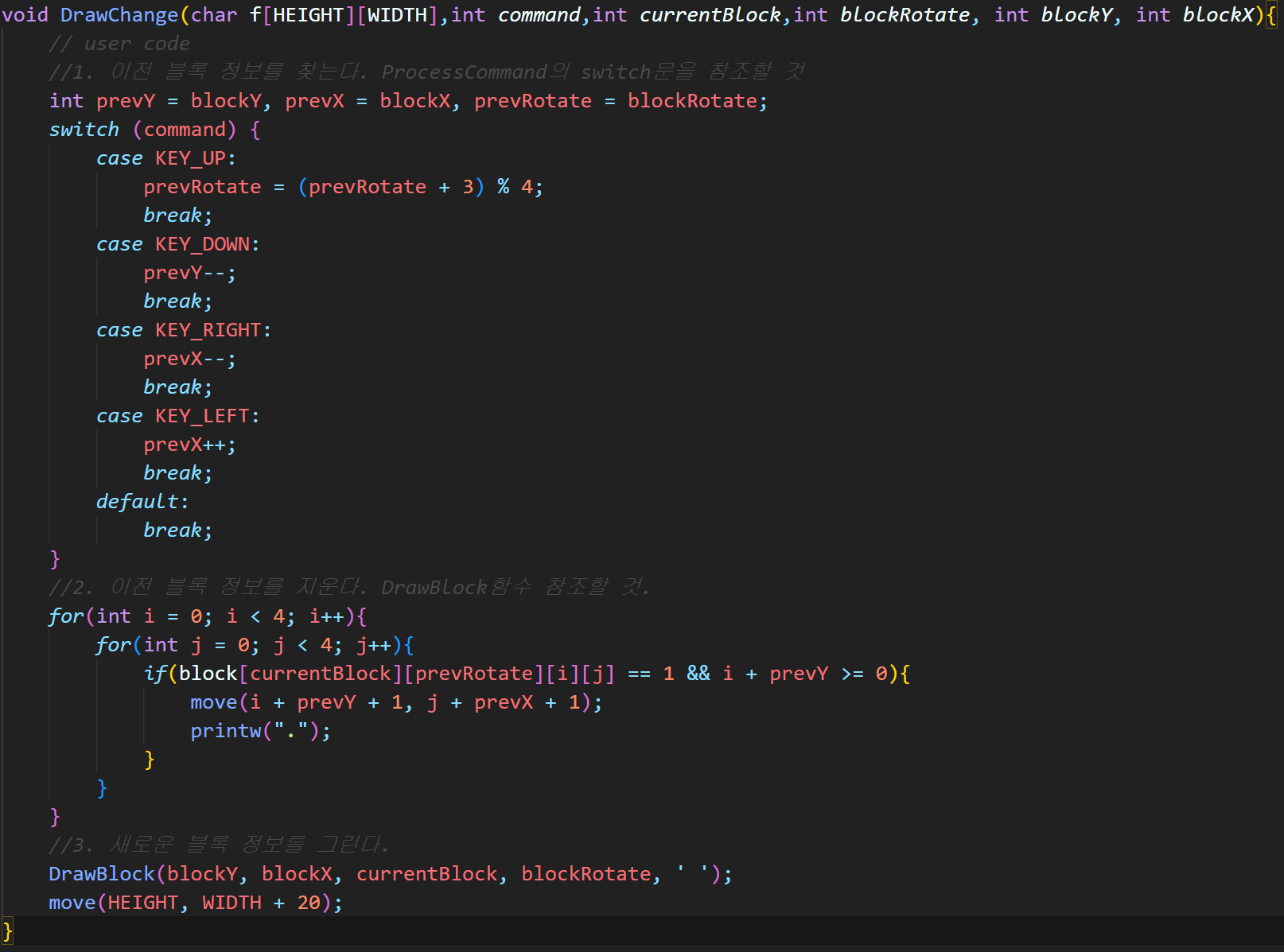
- CheckToMove() 함수에는 이중 for문이 사용되었으며, 각 반복문은 블록의 높이와 너비에 따라 순회한다. 따라서 블록의 크기를 n2이라 가정한다면 해당 함수의 시간 복잡도는 O(n2)이다. 즉, 블록의 크기에 따라 시간 복잡도가 선형적으로 증가하게 된다.

- 해당 함수는 추가적인 메모리를 사용하지 않고, 상수 개수의 변수만 사용하므로 공간 복잡도는 O(1)이다. 즉, 입력의 크기와 관계없이 일정한 공간을 사용한다.

***시간 복잡도 : O(n2)***

***공간 복잡도 : O(1)***

2) DrawChange() 함수



- pseudo code와 유사하지만 몇 가지 차이점이 존재한다. 먼저, pseudo code에서는 command 명령에 따라 블록을 이동 및 회전시킨다고 작성하였지만 실제 코드에서는 command에 따라 이전 블록의 위치 및 방향의 값을 적절하게 조정하였다.

- 또한, 이전 블록이 위치한 곳은 비워야 하므로 printw 함수를 호출하여 해당 위치에는 점(.)을 출력한다. 이렇게 함으로써 필드를 갱신할 수 있다. 마지막으로, 블록의 좌표를 이동시킨 후에는 새로운 블록을 해당 위치에 그리는 작업이 수행된다.

- 이러한 변경 사항으로 인해 함수의 시간 복잡도와 공간 복잡도는 앞서 설명한 CheckToMove() 함수와 동일하다. 따라서 DrawChange() 함수의 시간 복잡도는 O(n2)이며, 공간 복잡도는 O(1)이다.

***시간 복잡도 : O(n2)***

***공간 복잡도 : O(1)***

3) BlockDown() 함수



- pseudo code와 실제 코드의 흐름은 매우 유사하다. 실제 코드에서 CheckToMove() 함수를 사용하여 블록을 한 칸 아래로 이동할 수 있는지 확인한 후, 이동이 가능하다면 블록을 한 칸 아래로 이동시킨다. 그 후 DrawChange() 함수를 호출하여 필드를 업데이트하고 함수를 종료시킨다.

- 만약 블록이 필드의 가장 위에 위치하고 이동이 불가능한 경우 gameOver 변수를 1로 설정하는 것 역시 pseudo code와 동일하다. 그러나, 실제 코드에서는 점수 계산 및 필드를 업데이트하는 코드가 추가되었다.

- AddBlockToField() 함수를 통해 블록을 필드에 추가하고, 필드에서 완전한 라인을 지우고, 점수를 계산한다. 그리고 다음 블록을 세팅하고, 업데이트된 필드와 다음 블록을 화면에 그린다. 마지막으로 새로운 블록이 다음 블록이 되고, 게임 상태를 초기화한다.

- 해당 함수의 공간 복잡도는 이전 함수들과 동일하다. 그러나 이전 함수들과 달리 코드 내에서 반복문이 사용되지 않았기 때문에 BlockDown() 함수의 시간 복잡도는 O(1)이며, 공간 복잡도 또한 마찬가지로 O(1)이다.

***시간 복잡도 : O(1)***

***공간 복잡도 : O(1)***

4) AddBlockToField() 함수

텍스트, 소프트웨어, 멀티미디어 소프트웨어, 그래픽 소프트웨어이(가) 표시된 사진

자동 생성된 설명

- 해당 함수는 pseudo code와 동일한 코드 흐름을 갖는다. pseudo code에서 설명하였듯이 블록의 각 셀에 대해 해당 셀이 블록의 일부인지 확인한 후, 필드에서 그 위치에 1을 설정한다. 이 과정은 블록의 크기에 비례하여 수행되므로 블록의 크기가 n2이라면 시간 복잡도는 O(n2)이다.

- 공간 복잡도는 필드 배열의 크기와 관련이 있다. 필드 배열의 크기는 HEIGHT와 WIDTH로 주어지며, 이는 상수로 간주된다. 따라서 AddBlockToField() 함수의 공간 복잡도는 O(1)이다.

***시간 복잡도 : O(n2)***

***공간 복잡도 : O(1)***

5) DeleteLine() 함수

텍스트, 스크린샷, 디스플레이, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

- 해당 함수는 pseudo code와 거의 동일한 코드 흐름을 가지고 있지만, line이 꽉 차 있는 경우 코드의 차이가 존재한다. pseudo code에서는 한 줄을 삭제한 후에 다시 해당 줄을 확인하기 위해 인덱스를 조정하였지만, 그럴 필요가 없다. 실제 코드에서는 이중 for문을 통해 해당 line 위의 line들을 한 칸씩 아래로 이동시켰다. 이 과정을 통해 꽉 찬 line을 삭제할 수 있으며, 해당 줄을 다시 확인할 필요가 없게 된다.

- 시간 복잡도는 외부 반복문에서 O(HEIGHT)의 반복이 이루어지며, 내부 반복문에서는 O(WIDTH)의 반복이 이루어진다. 따라서 총 시간 복잡도는 O(HEIGHT \* WIDTH)이다.

- 공간 복잡도는 필드 배열의 크기에만 영향을 받으며, 상수 시간과 추가 공간을 사용하지 않는다. 따라서 총 공간 복잡도는 O(1)이 된다.

***시간 복잡도 : O(HEIGHT \* WIDTH)***

***공간 복잡도 : O(1)***

2. 테트리스 프로젝트 1주차 숙제 문제를 해결하기 위한 pseudo code를 기술하고, 시간 및 공간복잡도를 보이시오

1) 그림자 기능

(1) void DrawShadow(int y, int x, int blockID, int blockRotate){

while(블록이 아래로 움직일 수 있다면) {

y++;

}

DrawBlock(y, x, blockID, blockRotate, '/');

}

- DrawShadow() 함수는 최악의 경우, field를 HEIGHT만큼 탐색한다. 그리고 DrawBlock() 함수 호출은 상수 시간이 소요되므로 전체적으로 **O(HEIGHT)**의 시간 복잡도를 갖는다.

- 함수 내부에서 새로운 변수를 생성하지 않으며, 상수 크기의 메모리만 사용하므로 공간 복잡도는 **O(1)**이다.

(2) void DrawBlockWithFeatures(int y, int x, int blockID, int blockRotate){

1. 블록을 그린다. (DrawBlock 함수 사용)
2. 그림자를 그린다. (DrawShadow 함수 사용)

}

- DrawBlockWithFeatures() 함수는 DrawShadow() 함수와 DrawBlock() 함수를 호출한다. 함수의 호출만 이루어지기 때문에 **O(1)**의 시간 복잡도를 갖는다.

- DrawShadow() 함수와 마찬가지로 상수 크기의 메모리만 사용하기 때문에 **O(1)**의 공간 복잡도를 갖는다.

2) 2개의 블록 미리 보여주기

(1) void InitTetris(){

nextBlock[0]=rand()%7;

nextBlock[1]=rand()%7;

nextBlock[2]=rand()%7;

DrawBlockWithFeatures(blockY, blockX, nextBlock[0], blockRotate);

}

- 기존의 InitTetris() 함수에 다음 코드를 추가해주면 된다. 맨 마지막 DrawBlockWithFeatures() 함수는 기존의 DrawBlock() 함수를 대체한다.

- 해당 함수는 변수 초기화 및 함수의 호출로만 이루어져 있으므로 시간 복잡도는 **O(1)**이다.

- 상수 공간 및 field 배열만 사용되며, 추가적인 메모리를 사용하지 않기 때문에 공간 복잡도 또한 **O(1)**이다.

(2) void DrawNextBlock(int \*nextBlock){

int i, j;

for( i = 0; i < 4; i++ ){

move(4+i,WIDTH+13);

for( j = 0; j < 4; j++ ){

if( block[nextBlock[1]][0][i][j] == 1 ){

attron(A\_REVERSE);

printw(" ");

attroff(A\_REVERSE);

}

else printw(" ");

}

}

for( i = 0; i < 4; i++ ){

move(10+i,WIDTH+13);

for( j = 0; j < 4; j++ ){

if( block[nextBlock[2]][0][i][j] == 1 ){

attron(A\_REVERSE);

printw(" ");

attroff(A\_REVERSE);

}

else printw(" ");

}

}

}

- 기존의 함수에 커서 위치만 변경하여 동일한 작업을 한 번 더 반복한다.

- 해당 함수의 주요 연산은 이중 for문으로 이루어진다. 블록의 크기를 n2이라 가정했을 때, 각 루프는 블록의 크기만큼 반복되므로 전체 시간 복잡도는 **O(n2)**이다.

- 해당 함수에서 추가적으로 사용되는 공간은 상수 개수의 변수인 i와 j이다. 또한 nextBlock 배열을 사용하며, 이 배열은 integer 배열이지만 크기가 고정되어 있으므로 상수 공간이다. 따라서 전체 공간 복잡도는 **O(1)**이다.

3) 닿은 면적만큼 score 증가하기

(1)

int AddBlockToField(char f[HEIGHT][WIDTH],int currentBlock,int blockRotate, int blockY, int blockX){

// user code

int touched = 0;

//Block이 추가된 영역의 필드값을 바꾼다.

for (int i = 0; i < BLOCK\_HEIGHT; i++) {

for (int j = 0; j < BLOCK\_WIDTH; j++)

if(block[currentBlock][blockRotate][i][j] == 1) {

f[blockY + i][blockX + j] = 1;

if(block의 y좌표가 field의 가장 아랫부분과 맞닿거나 바로 아래 필드가 채워져 있다면)

touched++;

}

}

return touched \* 10;

}

- touched 변수를 생성하고, 각 블록의 셀마다 조건을 만족하면 해당 값을 업데이트한다. 마지막에 해당 변수의 값을 반환한다. touched 변수의 추가 외에는 기존 함수와 동일한 코드를 갖기 때문에 동일한 시간 복잡도 및 동일한 공간 복잡도를 갖는다.

**- 시간 복잡도: O(n2)**

**- 공간 복잡도: O(1)**

(3) void BlockDown(int sig){

score += AddBlockToField(field, nextBlock[0], blockRotate, blockY, blockX);

nextBlock[0] = nextBlock[1];

nextBlock[1] = nextBlock[2];

nextBlock[2] = rand() % 7;

}

- 기존의 함수에 AddBlockToField() 함수를 이용한 score 계산이 추가되었으며, nextBlock 배열의 값을 갱신하는 부분이 추가되었다.

- 그 외의 코드는 동일하기 때문에 기존의 함수와 동일한 시간 복잡도 및 공간 복잡도를 갖는다.

**- 시간 복잡도: O(1)**

**- 공간 복잡도: O(1)**