6주차 결과보고서

전공: 수학과,컴퓨터공학과 학년: 3학년 학번: 20181256 이름: 김도현

1.

들어가기에 앞서 해당 세부 코드 내용은 코드 자체에 주석(//)을 달아 설명하였다. Pseudo code와 실습의 코드 변화, 공간, 시간복잡도만 따로 적어주었다.

1-1. CheckToMove

Pseudo code이다.

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

{

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

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

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

//블록이 해당위치에 있을 때 (4x4 중)

i+blockY >=HEIGHT, j+blockX >=WIDTH, j+blockX<0,f[y][x]==1 => return 0

//원하는 위치의 y,x 값이 필드보다 크거나 작거나 이미 필드가 차있다면 0 return

return 1 //아니면 1을 return 즉 블록이 다른 위치로 바뀔 수 있음을 의미

}

실제 코드이다.

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

//블록이 회전이나 이동이 가능한지확인

// user code

for(int i=0;i<BLOCK\_HEIGHT;i++) //블록의 HEIGHT 만큼

{

for(int j=0;j<BLOCK\_WIDTH;j++) //블록의 WIDTH 만큼

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

int y=i+blockY; //움직이고자 하는 y 값이 field 범위?

int x=j+blockX; //움직이고자 하는 x 값이 field 범위?

if(y>=HEIGHT) return 0;//y 가 범위 넘으면 return 0

if(x>=WIDTH) return 0; //x 가 범위 넘으면 return 0

if(x<0) return 0; //x가 범위 넘으면 return 0

if(f[y][x]==1) return 0; //이미 f 차있으면 return 0

}

}

}

return 1; //checktomove 가능

}

여기서는 블록이 회전이나 이동이 가능한지 확인하는 함수이다. 블록이 있는 곳에서 각 y, x의 범위를 확인하여 가능하지 않으면 0을 Return 하고 가능하다면 1을 return 한다. 이때 pseudo code에서 어떤 범위에서 return을 할 것인지 적어줬지만 정확한 코드를 작성하지 않고 전체적인 흐름만 작성하였다. 이를 세부화하여 해당사항마다 return을 따로 적어주었다. 처음 2중 for문은 이전 코드의 흐름을 보고 적었지만 실습을 진행하면서 4\*4 공간 16개 중 1개씩 확인하는 해당 블록의 위치를 알리는 코드임을 정확히 인지하게 되었다. 나머지는 실습 때 공부한 그대로 작성하니 코드가 돌아감을 확인할 수 있었다.

여기서 2중 for문이 돌아가기 때문에 최대로 걸리는 시점인 for문이 2번 돌고 move가 가능하다고 판단하는 return 1일 때이다. 이때 시간 복잡도는 O(BLOCK\_WIDTH\*BLOCK\_HEIGHT) 가 된다. 또한 단순 for문 및 i,j 변수, input 참조값만 사용하고 변수 추가 메모리 크기변화가 없기 때문에 공간 복잡도는 O(1)로 생각할 수 있다.

1-2. DrawChange

Pseudo code 이다.

void DrawChange(char f[HEIGHT][WIDTH],int command, int currentBlock, int blockRotate, int blockY, int blockX)

{

switch(command)

case KEY\_UP: //key\_up이면 이전은 rotate가 시계방향을 의미

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

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

if(block[currentBlock][prevblockRotate][i][j]==1 && i+blockY>=0)

move(i+blockY+1,j+blockX+1)

printw(“.”) //이전 블록을 지우기

break

case KEY\_DOWN: //key\_down이면 이전은 y-1 위치를 의미

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

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

if(block[currentBlock][blockRotate][i][j]==1 && i+(prevY)>=0)

move(i+prevY+1, j+blockX+1)

printw(“.”) //이전 블록 지우기

break

case KEY\_LEFT: //key\_left이면 이전은 x+1 위치를 의미

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

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

if(block[currentBlock][blockRotate][i][j]==1 && i+blockY>=0)

move(i+blockY+1, j+prevX+1)

printw(“.”) //이전 블록 지우기

case KEY\_RIGHT: //key\_right이면 이전은 x-1 위치를 의미

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

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

if(block[currentBlock][blockRotate][i][j]==1 && i+blockY>=0)

move(i+blockY+1, prevX+1)

printw(“.”) //이전 블록 지우기

break

DrawBlock(blockY,blockX,currentBlock, blockRotate, “ ”) //새로운 블록 그리기

move(HEIGHT, WIDTH+10) //커서를 필드 밖으로 이동

}

실제 코드이다.

void DrawChange(char f[HEIGHT][WIDTH],int command,int currentBlock,int blockRotate, int blockY, int blockX){

// user code

//1. 이전 블록 정보를 찾는다. ProcessCommand의 switch문을 참조할 것

switch(command)

{

case KEY\_UP:for(int i=0;i<BLOCK\_HEIGHT;i++){ //key\_up 의 이전인 (blockroate+3)%4 를 통해 이전 정보 찾고 지우기

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

{

if(block[currentBlock][(blockRotate+3)%4][i][j]==1&&i+blockY>=0) //블록 찾고 위치 적절한지 확인한다음

{move(i+blockY+1,j+blockX+1);printw(".");} //printw를 통해 지우기

}

}

break;

case KEY\_DOWN:for(int i=0;i<BLOCK\_HEIGHT;i++){ //key\_down 의 이전인 blcokY-1 지우기

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

{

if(block[currentBlock][blockRotate][i][j]==1&&i+blockY-1>=0)

{move(i+blockY,j+blockX+1);printw(".");}

}

}

break;

case KEY\_LEFT:for(int i=0;i<BLOCK\_HEIGHT;i++){ //key\_left 의 이전인 blockX+1 지우기

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

{

if(block[currentBlock][blockRotate][i][j]==1&&i+blockY>=0)

{move(i+blockY+1,j+blockX+2);printw(".");}

}

}

break;

case KEY\_RIGHT:for(int i=0;i<BLOCK\_HEIGHT;i++){ //key\_right 의 이전버전인 blockX-1 지우기

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

{

if(block[currentBlock][blockRotate][i][j]==1&&i+blockY>=0)

{move(i+blockY+1,j+blockX);printw(".");}

}

}

break;

default: break;

}

//2. 이전 블록 정보를 지운다. DrawBlock함수 참조할 것.

DrawBlock(blockY,blockX,currentBlock,blockRotate, ' ');

//3. 새로운 블록 정보를 그린다.

move(HEIGHT,WIDTH+10); //커서를 밖으로 이동시켜주기

}

Command 및 초당 block이 떨어질 때 이전 block을 지우고 바뀐 block을 새로 그리는 함수이다.

처음 코드를 구상할 때 이전 블록을 가져오는 곳에서 어떻게 해야 이전 블록의 정보를 찾을 지 잘 몰랐다. 하지만 실습을 진행하면서 blockRotate가 바뀌는 곳에서는 blockRotate가 4개가 있는데 그중 1씩 증가하니 그 이전은 +3을 하고 (mod 4)를 하면 되는 것을 알게 되었다. 그래서 %4 나머지가 이전 블록의 blockrotate 이다. 또한 이전 블록의 Y, X 의 위치 역시 그냥 prevY, prevX 이렇게 적었는데 쉽게 생각해 Y값이 증가했다면 이전 블록의 Y 값은 Y-1, X값이 증가했다면 이전 블록의 X 값은 X-1, X값이 감소했다면 이전 블록 X 값은 X+1 로 생각하고 코드를 구성하면 되었다. 이를 각 case 문에서 block을 지우는 이중 for문을 이용해 지워준다. 또한 혹시 모를 에러를 대비해 default 값은 그냥 break를 줬는데 아마 안 적어도 정상 작동 할 것으로 보인다.

이후 새로운 블록을 그리고 커서를 다시 밖으로 옮기는 코드는 구상 그대로 작성하였다.

여기서 각 함수가 실행될때마다 각 case문 1개만 실행되고 여기 안에서는 2중 for 문이 돌아간다. 여기서는 O(BLOCK\_HEIGHT\*BLOCK\_WIDTH) 이다. DrawBlock 역시 2중 for문을 돌면서 block을 그리기 때문에 O(BLOCK\_HEIGHT\*BLOCKWIDTH)만큼의 시간이 소요된다. 나머지는 순차적인 흐름으로 함수가 실행되어 O(1)이다.

즉 최종 시간 복잡도는 O(BLOCK\_HEIGHT\*BLOCK\_WIDTH) 이고 또한 단순 for문 및 i,j 변수, input 참조값만 사용하고 변수 추가 메모리 크기변화가 없기 때문에 공간 복잡도는 O(1)로 생각할 수 있다.

1-3.

BlockDown

Pseudo 코드이다.

int BlockDown(int sig)

{

if(CheckToMove(f, nextBlock[0], blockRotate, blockY+1,blockX)) //checktomove로 blockY+1일 때 블록 한칸 내릴 수 있는지

blockY=blockY+1; //blockY값 1 추가

DrawChange(f, KEY\_DOWN, currentBlock, blockRotate, blockY,blockX) //key\_down으로 블록 그림

else

if (blockY==-1) gameOver=1 //블록 못 내리고 blocky==-1 이면 gameover

AddBlockToField(f, nextBlock[0],blockRotate, blockY,blockX) //아니면 필드에 블록 추가

nextBlock[0]=nextBlock[1] //블록 초기화 과정

nextBlock[1]=rand()&7;

blockRotate=0;blockY=-1,blockX=(WIDTH/2)-2

score=score+DeleteLine(f) //deleteline으로 꽉찬라인 지우고 score에 점수 추가

printScore(score) //스코어 출력

DrawNextBlock(nextBlock) //다음 블록을 해당 박스에 그리기

DrawField() //필드 업데이트

}

실제 코드이다.

void BlockDown(int sig){

// user code

if(CheckToMove(field, nextBlock[0],blockRotate,blockY+1,blockX))//checktomove를 통해 nextBlock[0] 블록을 밑으로 내릴 수 있는지 확인

{

blockY=blockY+1;// blockY 값을 하나 증가시킴

DrawChange(field,KEY\_DOWN, nextBlock[0],blockRotate,blockY,blockX); //blockY 값을 하나 증가시키고 drawchange로 그림

//강의자료 p26-27의 플로우차트를 참고한다.

}

else{

if(blockY==-1) gameOver=1; //blockY==-1 이면 더이상 떨어질 곳이 없어 게임 종료

else{

AddBlockToField(field, nextBlock[0],blockRotate,blockY,blockX); //field에 블록 추가

nextBlock[0]=nextBlock[1]; //nextblock[0] 을 다음 블록으로 설정

nextBlock[1]=rand()%7; //새로운 블록 추가

blockRotate=0; //기존의 block 정보들 초기화

blockY=-1; //초기화

blockX=WIDTH/2-2;// 초기화

score=score+DeleteLine(field); //score 추가 하기 위해 deleteline 호출

PrintScore(score); //점수 출력

DrawNextBlock(nextBlock); //다음 블록 해당 박스에 그리기

}

DrawField(); //추가된 Field 다시 그리기

DrawBlock(blockY,blockX,nextBlock[0],blockRotate, ' '); //다시 블록을 첫 시작에서 하도록

}

timed\_out=0; //시간 다시 0

block을 초당 한칸씩 내릴려고 할때 시행되는 함수이다.

해당 코드 구상시에는 ppt의 flow chart 흐름대로 따라서 코드를 작성하였다. 그래서 잘 적었다고 생각했는데 실제 코딩해보니 에러가 발생하였다. 이유는 마지막에 timed\_out=0 으로 초기화를 하지 않았기 때문이다. Int sig를 input으로 받아와 1초마다 이 코드가 독자적으로 실행이 되는데 timed\_out이 바뀌기 때문에 함수 실행이 끝난 후 다시 timed\_out=0 으로 하여 다시 실행 가능하도록 만들어줘야 한다. 또한 마지막 3번 째 줄인 DrawField()로 새로 Field를 그린 이후 DrawBlock으로 위에서 떨어질 블록을 새로 그리지 않았더니 게임 상 블록 생성이 가장 위에서 생성되지 않고 약간 아래서 만들어져 떨어졌다. 이를 해결하고자 함수가 끝나기 전 미리 블록을 생성했다. 그리고 흐름상 if 문으로 다 묶었는데 실제로는 blockY==-1( 블록이 떨어질 곳이 없어 게임종료) 인지 아닌 지로 나누고 이후 DrawField 부터는 상위 if else문으로 넣는게 좋을 것 같다고 생각했다.

여기선 block을 한칸 씩 내리는 함수인데 첫 if 문에서 CheckToMove의 O(BLOCK\_HEIGHT\*BLOCK\_WIDTH)가 있고 true 이면 DrawChange가 실행되면 위에서 설명했다시피 O(BLOCK\_HEIGHT\*BLOCK\_WIDTH) 이고 AddBlockToField 함수로 들어갈 시 이 역시 2중 for문으로 인해 이때 DrawField나 DeleteLine은 2중 for문이 O(HEIGHT\*WIDTH), O(HEIGHT\*WIDTH\*삭제줄위치)로 돌게 된다. 그래서 최종 시간복잡도는 O(HEIGHT\*WIDTH\*삭제줄위치) 로 생각하면 된다 또한, input 참조값만 사용하고 변수 추가 메모리 크기변화가 없기 때문에 공간 복잡도는 O(1)로 생각할 수 있다.

1-4.

AddBlockToField

Pseudo 코드이다.

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

{

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

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

if(Block[currentBlock][blockRotate][i][j]==1) //해당 위치에 블록이 있을때(4x4)

field[blockY+i][blockX+j]=1 //필드에 추가(blockY+i,blockX+j)

}

실제 코드이다.

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

// user code

for(int i=0;i<BLOCK\_HEIGHT;i++) //블록의 HEIGHT 만큼

{

for(int j=0;j<BLOCK\_WIDTH;j++) //블록의 WIDTH 만큼

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

f[blockY+i][blockX+j]=1; //필드에 해당 공간을 채운다

}

}

}

}

블록을 실제 Field 에 추가하는 AddBlockToField 는 ppt의 흐름대로 따라간다면 쉽게 작성이 가능하다. 때문에 코드를 추가로 작성하지 않았다.

여기서 어느곳에 block이 위치하는지 찾는 2중 for문이 있어 시간 복잡도는 O(BLOCK\_HEIGHT\*BLOCK\_WIDTH) 이고 단순 for문 및 i,j 변수, input 참조값만 사용하고 변수 추가 메모리 크기변화가 없기 때문에 공간 복잡도는 O(1)로 생각할 수 있다.

1-5. DeleteLine

Pseudo 코드이다.

int DeleteLine(char f[HEIGHT][WIDTH])

{

bool checkline=false //flag checkline

int completedLine=0 //꽉 찬 line 개수 변수

for(int i=0;i<HEIGHT<i++)

checkline=true //checkline=true로 두고 false가 나오는지 확인

for(int j=0;j<WIDTH;j++)

if f[i][j]==0 //필드 비었으면

checkline=false //checkline false

if(checkline) //checkline으로 꽉 찬 line 있으면

for(int k=i;k>=1;k--) //해당 y 축 그 위로 모든 블록들을 하나 아래로 내림

for(int s=0;s<WIDTH;s++)

f[k][s]=f[k-1][s] //x축은 그대로 내려옴

completedLine++ //line 1추가

return completedLine\*completedLine\*100 //점수는 (꽉 찬 line 제곱 )\* 100

}

실제 코드이다.

int DeleteLine(char f[HEIGHT][WIDTH]){

// user code

bool checkline=false;

int completedline=0;

for(int i=0;i<HEIGHT;i++)

{

checkline=true;

for(int j=0;j<WIDTH;j++)

{if(f[i][j]==1) //해당 필드의 [i][j]가 채워져 있으면 checkline=true, 아니면 checkline=false 하고 break; 다음 for문 돌아감

{checkline=true;}

else{checkline=false; break;}

}

if(checkline){ //checkline=true 이면

completedline++; //완성줄 1개 추가

for(int k=i;k>0;k--) //i 행 기준 그 위의 모든 블록들을 하나씩 내린다

{for(int s=0;s<WIDTH;s++){ //열은 그대로 한다

f[k][s]=f[k-1][s]; //field 를 아래로 한칸 씩 내린다

} }

}

}

//1. 필드를 탐색하여, 꽉 찬 구간이 있는지 탐색한다.

//2. 꽉 찬 구간이 있으면 해당 구간을 지운다. 즉, 해당 구간으로 필드값을 한칸씩 내린다.

int scored=completedline\*completedline\*100;

return scored;

}

완성된 줄이 있으면 그 줄을 찾고 해당 줄을 지우고 score 계산 후 return 하는 함수이다. Pseudo 문을 작성할 때 f[i][j]==0일때를 if 조건문으로 넣어 true 이면 그냥 checkline=false로 만들었다. 하지만 생각해보니 그 줄이 비었다는 의미이기 때문에 더이상 for문을 돌 필요가 없었다. 그래서 break 를 추가하여 f[i][j]=0 이면 가장 내부의 for문을 나갔다. 그리고 코드를 좀 더 확실히 하기 위해서 실제 코드에서는 f[i][j]==1 을 if 문의 조건으로 넣었다. completedLine을 언제 추가하는지 위치 변경과 같은 사소한 디테일 말고 나머지는 큰 차이가 없다.

해당 함수 역시 전체 필드에서 차 있는 부분을 확인하는 2중 for문(field의 Height,Width) 이 돌아가는데 checkline이 true 라면 해당 중을 지우는 과정이 추가된다. 여기서는 2번째 for 문안에 또 for문이 존재한다. O(삭제 줄 위치\*WIDTH). 나머지 코드는 흐름대로 따라가서 O(1)이다. 그래서 시간 복잡도는 O(HEIGHT\*(WIDTH+(삭제 줄 위치\*WIDTH))~>O(HEIGHT\*WIDTH\*(삭제 줄 위치)) 로 생각해 볼 수 있다. 또한 단순히 for문에 사용되는 변수 및 i,j,,k,s,checkline, completedline, scored 변수, input 참조값만 사용하고 변수 추가 메모리 크기변화가 없기 때문에 공간 복잡도는 O(1)로 생각할 수 있다.

2. 코드 및 어떤 함수인지에 대한 설명은 pseudo code에 기술하였다.

2-1. 그림자 기능

현재 block을 그리는 DrawBlock 는 char tile input을 받기 때문에 pseudo 코드를 생각할 때는 따로 추가 수정사항은 지금으로써는 없어보인다.

즉 시간복잡도 O(BLOCK\_HEIGHT\*BLOCK\*WIDTH), 공간복잡도는 O(1) 이다.

DrawShadow(int y, int x, int blockID, int blockRotate)

int shadow\_y=y //shadow\_y 에 y 값 저장

while(1)

if(CheckToMove(field, nextBlock[0], blockRotate,shadow\_y+1,x)

//해당 블록이 어디까지 아래로 내려갈 수 있는지 checkToMove로 확인하는 구간

shadow\_y=shadow\_y+1 //가능하다면 shawdow\_y를 한칸 내린다

else break; //아니면 for 문을 나온다.

DrawBlock(shawdow\_y, x, blockRotate, ‘/’) //가장 아래로 내렸을 때 가능한 부분에 ‘/’ 로 블록을 칠해준다.

실제 코드

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

// user code

int shadow\_y=y+1; //shadow\_y로 현재 y좌표 다음 y좌표를 둔다. 현재 위치에서 쭉 내려 그림자를 확인하기 때문에 x좌표는 신경안쓴다.

while(1){ //계속해서 break 나오기 전까지 반복한다

if(CheckToMove(field,blockID,blockRotate,shadow\_y,x)) //계속 y값을 하나씩 내릴 수 있는지 확인해본다,

shadow\_y=shadow\_y+1; //맞다면 shawdow\_y를 하나 증가 시켜 다음 checktoMove에 들어가도록 +1 한다.

else break; //더이상 내려갈 수 없다면 그 곳에 그림자가 생기면 될 것이다.

}

DrawBlock(shadow\_y-1,x,blockID,blockRotate,'/'); //drawBlock을 호출하여 (shadow\_y, x)위치에 '/' 로 그림자블록을 그린다.

}

그림자를 그려주는 DrawShadow 함수에 대한 내용이다.

Drawshadow의 while 반복문 시간 복잡도는 일반적으로 while문에서 break가 걸리는 최대 시점이 Height 이기 때문에 O(height) 로 예상할 수 있다. While문 안의 CheckToMove는 O(BLOCK\_HEIGHT\*BLOCK\_WIDTH), DrawBlock역시 O(BLOCK\_HEIGHT\*BLOCK\_WIDTH)이다. 최종적으로 시간 복잡도는 O(BLOCK\_HEIGHT\*BLOCK\_WIDTH\*HEIGHT)로 예상해본다. 또한 shadow\_y 변수, input 참조값만 사용하고 변수 추가 메모리 크기변화가 없기 때문에 공간 복잡도는 O(1)로 생각할 수 있다.

DrawBlockWithFeatures(int y, int x, int blockID, blockRotate)

DrawBlock(y,x,blockID, blockRotate, ‘ ‘) //drawblock 호출 블록 그림

DrawShadow(y,x,blockID, blockRotate) //drawshadow 호출하여 그림자 그림

실제코드

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

DrawBlock(y,x,blockID, blockRotate,' ');// block과 shadow를 그려준다.

DrawShadow(y,x,blockID, blockRotate);

}

해당 함수는 따로 주어지지 않았기 때문에 tetris.h 에 추가하여 함수를 작성해야 할 것이다.

DrawBlockWithFeatures 는 DrawBlock, DrawShadow 함수를 호출하여 블록과 그림자를 그려주는 호출 함수이다. 시간복잡도는 각 함수에 따라간다고 볼 수 있는데 O(BLOCK\_HEIGHT\*BLOCK\_WIDTH+BLOCK\_HEIGHT\*BLOCK\_WIDTH\*HEIGHT)라 생각할 수 있다. 편히 O(BLOCK\_HEIGHT\*BLOCK\_WIDTH\*HEIGHT)라 보면 된다. input 참조값만 사용하고 변수 추가 메모리 크기변화가 없기 때문에 공간 복잡도는 O(1)로 생각할 수 있다.

이후 기존 DrawBlock(blockY, blockX, nextBlock[0],blockRotate,’ ’); 로 그렸던 것들을 (예를 들어 DrawChange나 initTetris 등)DrawBlockWithFeatures(int y, int x, int blockID, blockRotate) 로 모두 변경한다. 이것으로 인해 공간복잡도나 시간복잡도는 변하지 않는다.

하지만 ppt에서는 설명하지 않았지만 그림자를 지우는 과정도 포함이 되어야 한다. 때문에 DrawChange에서 이전 블록을 지우듯이 이전 그림자도 같이 지워주는 과정을 포함했다. 때문에 각 case 문에서 가장 안쪽 for 문 에서 while 문을 추가하여 위의 drawshadow 처럼 이전 위치의 블록이 그림자를 가지는 위치를 찾아 그곳을 지워준다.

Case KEY\_UP: for(int i=0;i<BLOCK\_HEIGHT;i++){

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

{

if(block[currentBlock][(blockRotate+3)%4][i][j]==1&&i+blockY>=0) //블록 찾고 위치 적절한지 확인한다음

{move(i+blockY+1,j+blockX+1);printw(".");

int shadow\_y=blockY+1; //shadow\_y로 현재 y좌표 다음 y좌표를 둔다. 현재 위치에서 쭉 내려 그림자를 확인하기 때문에 x좌표는 신경안쓴다.

while(1){ //계속해서 break 나오기 전까지 반복한다

if(CheckToMove(f,currentBlock,(blockRotate+3)%4,shadow\_y,blockX)) //계속 y값을 하나씩 내릴 수 있는지 확인해본다,

shadow\_y=shadow\_y+1; //맞다면 shawdow\_y를 하나 증가 시켜 다음 checktoMove에 들어가도록 +1 한다.

else break; //더이상 내려갈 수 없다면 그 곳에 그림자가 생기면 될 것이다.

}

move(i+shadow\_y,j+blockX+1);printw("."); //그림자도 같이 지워준다

} //printw를 통해 지우기

}

나머지 생략.. 똑같이 해준다.

}

이 코드의 추가로 DrawChange 는 이중 for문 안에 또 while 문에서 CheckToMove가 있어 시간 복잡도는 O(BLOCK\_HEIGHT\*BLOCK\_WIDTH\*(BLOCK\_HEIGHT\*BLOCK\_WIDTH)\*HEIGHT) 가 된다. 또한 단순 for문 및 i,j 변수, input 참조값만 사용하고 변수 추가 메모리 크기변화가 없기 때문에 공간 복잡도는 O(1)로 생각할 수 있다.

2-2. 2개 블록 미리 보여주기

InitTetris()

nextBlock[2]=rand()%7 // 다음 다음 Block을 보일 수 있도록 한다. 이때 주의해야할 점은 Block\_num이 tetris.c에서 2로 되어있을 건데 배열 3개를 받기 위해 3으로 수정할 것이다.

DrawBlock(blockY, blockX, nextBlock[0],blockRotate, ‘ ‘)=>

DrawBlockWidthFeatures(blockY, blockX, nextBlock[0],blockRoate) 로 처음 그렸을 때 그림자 역시 받아올 수 있도록 수정할 것이다.

시간 복잡도는 한줄이 추가되어 O(1)이여서 바뀌지 않는다. 그대로 O(HEIGHT\*WIDTH) 이고 공간 복잡도도 추가된 것은 O(1)로 기존과 비교해 바뀌지 않는다. 그대로 O(HEIGHT+WIDTH)이다.

DrawNextBlock(int \*nextBlock)

DrawNextBlock(int \*nextBlock)

nextBlock[1] 관련 코드…

//nextBlock[2] 도 동일하게 해줌

For( i=0;i<4;i++)

move(? ;WIDTH+13) //커서를 y축으로 몇번째 칸으로 옮겨야 할지는 아직 못정했다.

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

if (block[nextBlock][2][0][i][j]==1) //nextBlock[2] 의 정보를 가져와 다다음 블록 역시 그릴수 있게 한다.

attron(A\_REVERSE); //문자색 반전하여 출력

printw(“ “);

attroff(A\_REVERSE); //문자색 반전하여 출력

else printw(“ “);

나머지는 nextblock[1] 과 동일하게 코드를 구성하면 될것이다.

그냥 nextBLock[2] 하나만 추가하기 때문에 시간 복잡도는 그대로 O(BLOCK\_HEIGHT\*BLOCK\_WIDTH) 이고 공간 복잡도 또한 그대로 O(1) 일 것이다.

DrawOutline()

Draw(3,WITDH+10,4,8)

DrawBox(10, WIDTH+10, 4, 8) //추가로 nextBlock이 그려질 Box 또한 같이 그린다.

move(17,WIDTH+10) //score 공간은 실습에 비해 약간 내려야한다

printw(“SCORE”)

DrawBox(18,WIDTH+10,1,8)

그리고 원래 score 있던 자리를 내려야 한다. 이는 코드를 실제 구현하면서 알아볼 것이다.

순차적으로 코드가 흘러가고 추가 및 변경으로 인해 공간, 시간복잡도 변화는 없다. 그대로 DrawBox에서 시간복잡도 O(width 또는 height), 변수의 크기 변화도 없어 공간복잡도는 O(1) 이다.

BlockDown(int sig)

nextBlock[1]=nextBlock[2] //nextBlock이 1개 더 늘었기 때문에 nextBlock1을 2로 지정하고

nextBlock[2]= rand()%7 //nextBlock2는 새로 랜덤하게 블럭을 할당한다.

추가 변화된 사항이 복잡도에 영향을 끼치지 않아 시간복잡도는 그대로 O(BLOCK\_HEIGHT\*BLOCK\_WIDTH\*삭제줄위치), 공간복잡도 O(1) 이다.

2-3. score 증가 수정

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

Int touched=0 //터치하는 부분을 세는 변수 추가

For

For

If ()

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

if(blockY+i==HEIGHT-1) touched++ //만약 현재의 Y값이 Height 보다 하나 작은 즉 마지막 줄에 닿게 된다면 해당 touched를 1 늘린다.

else if(f[blocky+i+1][blockX+j]!=0) counted++

//만약 바닥면이 아닌 필드에블록이 닿는 경우는 해당 블록의 하나 밑 필드가 비어 있지 않을경우에도 count 하나 늘림

int touchedscore=touched\*10 //해당 touched의 10만큼 score를 늘린다

Return touchedscore;

Score 점수를 field와 touch한 만큼에 10을 곱해 return 해주는 것이 추가되었다.

여기서 원래 addblocktofield의 return 은 void 였다. 하지만 score int 값을 return 해야 하기 때문에 tetris.h 에서도 return 값을 void에서 return 으로 변경해야 할 것이다.

해당 함수는 2중 for문안에 if 조건문만 추가했기 때문에 시간복잡도는 그대로 O(BLOCK\_HEIGHT\*BLOCK\_WIDTH) 이고 touched , touchedscore를 변수로 추가했으나 한번 크기를 줬기 때문에 공간복잡도는 O(1) 이다.

BlockDown(int sig)

AddBlockToField(field, nextBlock[0], blockRoate, blockY, blockX) //줄 삭제

score=score+deleteLine(field) //줄 삭제

이렇게 변경한다.

Score=score+AddBlockToField(field, nextBlock[0], blockRoate, blockY, blockX)+DeleteLine(field)

// AddBlockTofield에서 touchedScore 받아온것을 더해준다.

Score를 deleteline과 AddblockTofield 에서 둘다 받아온다.

마찬가지로 touchedScore를 받아오는 것은 기존의 복잡도에 영향을 끼치지 않아 시간복잡도는 변하지 않는다. BlockDown에서 시간영향이 가장 큰 Deleteledline에서의 시간복잡도는 O(HEIGHT\*WIDTH\*(삭제 줄 위치)) 로 생각해 볼 수 있다. 공간복잡도 역시 그대로 변화없이 O(1) 이다.