8주차 결과보고서

전공: 아트&테크놀로지학과 학년: 4학년 학번: 20191048 이름: 김도솔

**1.**

**1) CheckToMove :** 의사코드의 각 줄을 명령어로 변환한 것 외에는 별다른 차이가 없었다. 루프의 반복 횟수는 상수인 4x4로 입력 크기에 관계없이 실행 시간이 일정하기 때문에 시간복잡도는 O(1)이다. 함수 내에서 추가적인 메모리 공간을 사용하지 않고, 입력 변수와 상수만을 이용하여 연산을 수행하기 때문에 입력 크기에 관계없이 일정한 공간을 사용한다. 따라서 공간복잡도는 O(1)이다.

***(pseudo code)***

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

i가 0에서 3까지

j가 0에서 3까지

block[currentBlock][blockRotate][i][j]==1이면

x = blockX+ j

y = blockY + i

블록을 나타내는 4×4 행렬의 각 요소의 실제 필드상의 y 좌표가 HEIGHT보다 크

거나 같으면

0을 반환

블록을 나타내는 4×4 행렬의 각 요소의 실제 필드상의 x 좌표가 0보다 작으면

0을 반환

블록을 나타내는 4×4 행렬의 각 요소의 실제 필드상의 x 좌표가 WIDTH보다 크

거나 같으면

0을 반환

블록을 놓으려고 하는 필드에 이미 블록이 쌓여져 있으면

0을 반환

1을 반환

***(실제 구현 함수)***

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

// user code

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

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

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

int x = blockX + j;

int y = blockY + i;

// 예상된 현재 블록이 필드 바깥으로 나갈 경우

if (y >= HEIGHT) return 0;

if (x < 0) return 0;

if (x >= WIDTH) return 0;

// 예상 현재 블록이 쌓여 있는 블록과 겹치는 경우

if (field[y][x] == 1) return 0;

}

}

}

return 1;

}

**2) DrawChange :** 이전 블록의 정보를 찾는 코드는 switch문으로 구현하였고, 화면에서 이전 블록을 지우는 코드는 DrawBlock 함수를 변형해 구현했다. 변경된 현재 블록을 그리는 코드는 DrawBlock 함수를 호출하여 완성했다. CheckToMove 함수와 동일하게 루프의 반복 횟수는 상수인 4x4로 입력 크기에 관계없이 실행 시간이 일정하기 때문에 시간복잡도는 O(1)이다. 또한 함수 내에서 추가적인 메모리 공간을 사용하지 않고, 입력 변수와 상수만을 이용하여 연산을 수행하기 때문에 입력 크기에 관계없이 일정한 공간을 사용한다. 따라서 공간복잡도는 O(1)이다.

***(pseudo code)***

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

현재 수행하게 될 command를 입력받는다

command == KEY\_UP이면 블록 회전 수 변경

command == KEY\_DOWN이면 블록의 Y좌표 -1

command == KEY\_LEFT면 블록의 X좌표 +1

command == KEY\_RIGHT면 블록의 X좌표 -1

화면에 그려진 기존 블록을 지운다

변경된 위치나 모양의 블록을 DrawBlock() 함수를 이용해 다시 그린다.

***(실제 구현 함수)***

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

// user code

int preBlock = currentBlock;

int preBlockRotate = blockRotate;

int preBlockY = blockY;

int preBlockX = blockX;

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

switch (command) {

case KEY\_UP:

preBlockRotate = (blockRotate + 3) % 4;

break;

case KEY\_DOWN:

preBlockY--;

break;

case KEY\_RIGHT:

preBlockX--;

break;

case KEY\_LEFT:

preBlockX++;

break;

default:

break;

}

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

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

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

if (block[preBlock][preBlockRotate][i][j] == 1 && i + preBlockY >= 0) {

move(i + preBlockY + 1, j + preBlockX + 1);

printw(".");

}

}

}

move(HEIGHT, WIDTH + 10);

//3. 새로운 블록 정보를 그린다.

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

}

**3) BlockDown :** 필드 및 블록 정보를 갱신해 다시 그리는 부분은 위에서 만든 DrawChange 함수를 호출해 구현하였고 라인을 검사해 점수를 계산하는 부분은 DeleteLine 함수를 호출해 구현하였다. 또한 if-else문이 끝난 후 timed\_out 변수를 0으로 설정해 블록이 한 칸씩 아래로 떨어질 수 있도록 하였다. 이 외에는 작성된 의사 코드 그대로 명령어로 변환하여 구현했다.

DeleteLine과 DrawField 함수가 각각 O(HEIGHT\*WIDTH)의 시간복잡도를 가지기 때문에 이 함수의 시간 복잡도도 O(HEIGHT\*WIDTH)이다. 입력 크기에 관계없이 일정한 공간을 사용하기에 공간복잡도는 O(1)이다.

***(pseudo code)***

void BlockDown(int sig)

블록을 한 칸 내릴 수 있으면 (CheckToMove() 함수 이용)

블록을 한 칸 내린다

필드 정보 및 블록 정보를 갱신한다

다시 그린다

블록을 한 칸 내릴 수 없을 때

블록의 y좌표가 -1이면

gameover를 1로 갱신한다

갱신된 정보를 다시 그린다

블록의 y좌표가 -1이 아니면

AddBlockToField() 함수를 사용하여 블록을 필드에 추가한다

필드 상에 빈 칸이 없는 라인이 있으면

라인을 삭제한다

제거된 라인의 수를 센다

점수를 계산해서 누적한다

다음 블록을 현재 블록으로 설정한다

각 블록에 대한 변수 초기화

다음 블록의 ID 새롭게 임의로 생성한다

필드 정보를 갱신하여 다시 그린다

***(실제 구현 함수)***

void BlockDown(int sig){

// user code

//강의자료 p26-27의 플로우차트를 참고한다.

if (CheckToMove(field, nextBlock[0], blockRotate, blockY + 1, blockX)) {

blockY++;

DrawChange(field, KEY\_DOWN, nextBlock[0], blockRotate, blockY, blockX);

}

else {

if (blockY == -1) {

gameOver = 1;

}

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

score += DeleteLine(field);

nextBlock[0] = nextBlock[1];

nextBlock[1] = rand() % 7;

DrawNextBlock(nextBlock);

blockRotate = 0;

blockY = -1;

blockX = WIDTH / 2 - 2;

PrintScore(score);

DrawField();

}

timed\_out = 0;

}

**4) AddBlockToField :** 의사 코드와 실제 구현 함수 간의 차이가 거의 없다. 고정된 크기의 루프를 반복하므로 실행 시간이 입력 크기에 영향을 받지 않아 시간복잡도는 O(1)이다. 또한 입력 크기에 관계없이 일정한 공간을 사용하기에 공간복잡도도 O(1)이다.

***(pseudo code)***

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

i가 0에서 3까지

j가 0에서 3까지

현재 블록이 채워져 있으면(currentBlock[i][j] == 1이면)

현재 테트리스 필드의 정보(field[blockY+i][blockX+j])를 1로 갱신

***(실제 구현 함수)***

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

// user code

//Block이 추가된 영역의 필드값을 바꾼다.

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

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

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

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

}

}

}

}

**5) DeleteLine :** 라인이 꽉 채워져 있는지 여부를 확인하기 위해 flag 변수를 도입해 구현했다. 라인에 빈칸이 있으면 flag를 0으로 설정하고 빈칸이 없으면 flag를 1로 설정했다. 이중 loop를 이용해 꽉 찬 라인이 있으면 해당 라인을 지우고 필드 정보를 한 줄씩 내리는 작업을 진행하였다.

이 함수의 시간복잡도는 O(HEIGHT \* WIDTH)이다. 함수는 필드의 모든 위치를 탐색하며, 각 행에 대해 꽉 찬 구간을 확인하는 작업을 수행한다. 따라서 전체 필드의 크기에 비례하여 실행 시간이 증가한다. 공간복잡도는 O(1)이다. 함수 내에서 추가적인 메모리 공간을 사용하지 않으며, 입력 변수인 필드 배열만을 사용하여 연산을 수행하기 때문에 입력 크기에 관계없이 일정한 공간을 사용한다.

***(pseudo code)***

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

i가 0에서 HEIGHT – 1 까지

j가 0에서 WIDTH - 1 까지

빈 칸이 있으면 다음 라인 체크

빈 칸이 없는 라인이 있으면

지워진 라인 수를 증가

라인을 테트리스 필드에서 삭제

지워진 라인 위의 필드 정보를 모두 한 줄씩 내려 갱신

지워진 라인 수를 사용해 점수를 계산

점수 반환

***(실제 구현 함수)***

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

// user code

//1. 필드를 탐색하여, 꽉 찬 구간이 있는지 탐색한다.

int line\_num = 0;

int score = 0;

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

int flag = 1;

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

if (f[i][j] == 0) flag = 0;

}

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

if (flag == 1) {

line\_num++;

for (int y = i; y > 0; y--) {

for (int x = 0; x < WIDTH; x++) {

f[y][x] = f[y - 1][x];

}

}

for (int x = 0; x < WIDTH; x++) {

f[0][x] = 0;

}

i--;

}

}

score = 100 \* (line\_num) \* (line\_num);

return score;

}

**2.**

**1) 그림자 기능**

**1-1) DrawBlock :** 시간복잡도와 공간복잡도는 O(1)이다.

void DrawBlock(int y, int x, int blockID,int blockRotate,char tile){

int i,j;

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

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

if(block[blockID][blockRotate][i][j]==1 && i+y>=0){

move(i+y+1,j+x+1);

attron(A\_REVERSE);

printw("%c",tile); // tile을 ‘/’로 설정

attroff(A\_REVERSE);

}

}

move(HEIGHT,WIDTH+10);

}

**1-2) DrawShadow :** 시간복잡도는 O(HEIGHT), 공간복잡도는 O(1)이다.

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

블록의 현재 y값에서 HEIGHT - 1 까지

그림자의 위치를 찾으면 (CheckToMove 함수 이용)

그림자 y값 저장 후 루프 탈출

DrawBlock(그림자 y값, x, blockID, blockRotate, ‘/’)

**1-3) DrawBlockWithFeatures :** 시간복잡도는 O(HEIGHT), 공간복잡도는 O(1)이다.

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

DrawShadow(y, x, blockID, blockRotate)

DrawBlock(y, x, blockID, blockRotate, ‘ ’)

**2) 2개의 블록 미리 보여주기**

**2-1) InitTetris :** 시간복잡도는 O(HEIGHT\*WIDTH)이고 공간복잡도는 O(1)이다.

void InitTetris() {

int i,j;

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

for(i=0;i<WIDTH;i++)

field[j][i]=0;

// 수정

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

nextBlock[i]=rand()%7;

}

(생략)

// 수정

DrawBlockWithFeatures(y, x, nextBlock[0], blockRotate);

(생략)

}

**2-2) DrawNextBlock :** 시간복잡도와 공간복잡도는 O(1)이다.

void DrawNextBlock(int \*nextBlock){

for (int k = 0; k < BLOCK\_NUM; k++) {

int i, j;

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

move(-2 + i + (k \* 6),WIDTH+13);

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

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

attron(A\_REVERSE);

printw(" ");

attroff(A\_REVERSE);

}

else printw(" ");

}

}

}

}

**2-3) BlockDown :** 시간복잡도는 O(HEIGHT\*WIDTH)이고 공간복잡도는 O(1)이다.

void BlockDown(int sig){

if (CheckToMove(field, nextBlock[0], blockRotate, blockY + 1, blockX)) {

(생략)

}

else {

(생략)

// 수정

for (int i = 0; i < BLOCK\_NUM - 1; i++) {

nextBlock[i] = nextBlock[i + 1];

}

nextBlock[BLOCK\_NUM - 1] = rand() % 7;

(생략)

}

timed\_out = 0;

}

**3) 닿은 면적만큼 score 증가하기**

**3-1) AddBlockToField :** 시간복잡도와 공간복잡도는 O(1)이다.

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

int touched = 0;

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

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

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

int y = blockY + i;

int x = blockX + j;

if (y >= HEIGHT) continue;;

if (x < 0) continue;

if (x >= WIDTH) continue;

f[y][x] = 1;

if(f[y + 1][x] || y + 1 == HEIGHT) touched++;

}

}

}

int score = touched \* 10;

return score

}

**3-2) BlockDown :** 시간복잡도는 O(HEIGHT\*WIDTH)이고 공간복잡도는 O(1)이다.

void BlockDown(int sig){

if (CheckToMove(field, nextBlock[0], blockRotate, blockY + 1, blockX)) {

(생략)

}

else {

(생략)

// 수정

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

if (blockY == -1) {

gameOver = 1;

}

score += DeleteLine(field);

(생략)

}

timed\_out = 0;

}