8주차 결과보고서

전공: 아트엔테크놀로지 학년: 3학년 학번: 20191172 이름: 함승우

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

1-1).

|  |
| --- |
| int CheckToMove(char f[HEIGHT][WIDTH],int currentBlock,int blockRotate, int blockY, int blockX)  //블록이 해당 위치로 이동하거나 회전할 수 있으면 1, 아니면 0을 return한다.  for block의 가로 길이  for block의 세로 길이  if 현재 block(현재 회전수, 현재 block)이 존재할 때  if 변화한 block의 x 및 y 좌표가 필드 내에 없을 때  return 0  else if 변화한 block의 좌표에 field가 1일 때  return 0  위가 아니라면 return 1 |

|  |
| --- |
| 261 int CheckToMove(char f[HEIGHT][WIDTH],int currentBlock,int blockRotate, int blockY, int blockX){  262 int i, j;  263 for(i=0; i <BLOCK\_WIDTH; i++)  264 {  265 for(j=0; j <BLOCK\_HEIGHT; j++)  266 {  267 if (block[currentBlock][blockRotate][i][j] == 1 ){  268 if( (i+blockY) >= HEIGHT)  269 return 0;  270 else if ((i + blockY) < 0)  271 return 0;  272 else if ((j + blockX) < 0)  273 return 0;  274 else if ((j + blockX) >= WIDTH)  275 return 0;  276 else if (f[i+blockY][j+blockX])  277 return 0;  278 }  279 }  280 }  281 return 1;  282  283 } |

위 함수는 이중 for문을 사용했기 때문에 시간 복잡도는 O(BLOCK\_WIDTH\*BLOCK\_HEIGHT)이다.

공간 복잡도는 입력된 파라미터들은 다 모두 고정된 크기를 갖기 때문에 공간 복잡도는 O(1)이라고 볼 수 있다.

1-2)

|  |
| --- |
| void DrawChange(char f[HEIGHT][WIDTH],int command,int currentBlock,int blockRotate, int blockY, int blockX)  //  ProcessCommand의 switch문을 참조하여 command 이전 블록 정보를 찾는다.  Switch(command)  1.방향키 위  이전 블록 정보 = (현재 블록의 회전수 -1)%4  2.방향키 아래  이전 블록 정보 = (현재 블록의 y좌표 - 1)  3. 방향키 왼쪽  이전 블록 정보 = (현재 블록의 x좌표 + 1)  4. 방향키 오른쪽  이전 블록 정보 = (현재 블록의 x좌표 - 1)  그 후 이전 블록 정보 부분에 ‘.’으로 채운다.  for 블록의 가로 길이  for 블록의 세로 길이  printw(“.”)  DrawBlock() 함수를 활용해서 그린다. |

|  |
| --- |
| 285 void DrawChange(char f[HEIGHT][WIDTH],int command,int currentBlock,int blockRotate, int blockY, int blockX){  286 // user code  287  288 //1. 이전 블록 정보를 찾는다. ProcessCommand의 switch문을 참조> 할 것  289 //2. 이전 블록 정보를 지운다. DrawBlock함수 참조할 것.  290 //3. 새로운 블록 정보를 그린다.  291  292 // 이전 블록 정보 찾는 과정, ProcessCommand의 switch문의 이전과정(-1)을  사용해서 이전 정보를 찾는다.  293 int pre\_blockRotate = blockRotate;  294 int pre\_blockY = blockY;  295 int pre\_blockX = blockX;  296  297 switch(command){  298 case KEY\_UP:  299 pre\_blockRotate = (blockRotate - 1)%4;  300 if (pre\_blockRotate < 0)  301 pre\_blockRotate = 3;  302 break;  303  304 case KEY\_DOWN:  305 pre\_blockY--;  306 break;  307  308 case KEY\_RIGHT:  309 pre\_blockX--;  310 break;  311  312 case KEY\_LEFT:  313 pre\_blockX++;  314 break;  315  316 default:  317 break;  318 }  319  320 int i,j;  321  322  323 for(i = 0; i < BLOCK\_WIDTH; i++){  324 for(j = 0; j < BLOCK\_HEIGHT; j++){  325 if((block[currentBlock][pre\_blockRotate][i][j] == 1) && (i +pre\_blockY > -1)){  326 move(i+pre\_blockY+1, j+pre\_blockX+1);  327 printw(".");  328  329 }  330 }  331 }  332  333  334  335 DrawBlock(blockY, blockX, currentBlock, blockRotate );  336 move(HEIGHT, WIDTH + 10);  337  338 } |

새로운 변수 pre\_blockRotate, pre\_blockY, pre\_blockX를 초기화해서 case문을 이용해서 이전 블록 정보들을 받는다. 그 후, 필드 위에 block이 있을 경우(line323 ~ line331) .으로 채우게 된다. 이 함수의 시간 복잡도는 line 323 ~ line 331의 함수 구문에 집중하면 된다, 이중 for문을 BLOCK\_WIDTH와 BLOCK\_HEIGHT로 구성하고 있기 때문에 시간 복잡도는 O(BLOCK\_WIDTH \* BLOCK\_HEIGHT)가 된다. 이 함수도 마찬가지로 입력된 변수 I, j, currentBlock, pre\_blockRotate, pre\_blockY, pre\_blockX 모두 고정된 크기를 갖기 떄문에 공간 복잡도는 O(1)이다.

1-3)

|  |
| --- |
| void BlockDown(int sig)  if 아래로 움직일 수 있다면  blocky 1 증가  else  gameover인 경우, block의 y좌표가 -1일 때  gameover  아니라면, 그 블록을 addblocktofield함수를 활용해서 그리고, 새로운 block을 설정하기 위해 초기화한다. initialize 함수 참조 |

|  |
| --- |
| 339 void BlockDown(int sig){  340 // user code  341 if(CheckToMove(field, nextBlock[0], blockRotate, blockY+1, bloc kX)){  342 blockY++;  343 DrawChange(field,KEY\_DOWN, nextBlock[0], blockRotate, b lockY, blockX);  344  345 }  346 else  347 {  348 if(blockY == -1)  349 gameOver = 1;  350  351 AddBlockToField(field, nextBlock[0], blockRotate, block Y, blockX);  352  353  354 score += DeleteLine(field);  355 nextBlock[0] = nextBlock[1];  356 nextBlock[1] = nextBlock[2];  357  358  359  360 //initialize location of current block)  361 // 점수 업데이트  362  363 blockRotate=0;  364 blockY=-1;  365 blockX=WIDTH/2-2;  366  367 DrawField();  368 DrawNextBlock(nextBlock);  369 PrintScore(score);  370 DrawBlock(blockY,blockX,nextBlock[0],blockRotate,' ');  371 }  372 timed\_out=0;  373 } |

block을 아래로 움직일 수 있다면 blockY를 증가시키는 식으로 진행을 하지만, block을 아래로 움직일 수 없는 상황에는 두 가지 경우가 있다. 첫 번째는 block의 y좌표가 -1으로 더 이상 내려갈 수 없는 상태를 의미한다. 이는 게임에서 gameover flag를 1로 올려 gameover를 시킨다. 두 번째 경우는 다른 블록이나 filed의 아래 경계에 위치한 경우로 자연스럽게 AddBlockTofield 함수를 이용해서 필드에 추가하면 된다. 그 이후 다음 block을 blockDown하기 전에 초기화를 시키는 과정을 거친다. 이중 for문을 BLOCK\_WIDTH와 BLOCK\_HEIGHT로 구성하고 있기 때문에 시간 복잡도는 O(BLOCK\_WIDTH \* BLOCK\_HEIGHT)가 된다. 이 함수도 마찬가지로 입력된 변수 I, j, blockX 등 변수들이 모두 고정된 크기를 갖기 떄문에 공간 복잡도는 O(1)이다.

1-4)

|  |
| --- |
| void AddBlockToField(char f[HEIGHT][WIDTH],int currentBlock,int blockRo tate, int blockY, int blockX)  //블록이 4X4이기 때문에  For 4  For 4  현재 block이 있다면  이 block이 추가된 영역의 필드를 1로 바꿔준다. |

|  |
| --- |
| 375 void AddBlockToField(char f[HEIGHT][WIDTH],int currentBlock,int blockRot ate, int blockY, int blockX){  376 // user code  377  378  379 //Block이 추가된 영역의 필드값을 바꾼다.  380 int i, j;  381 for(i = 0; i < 4; i++){  382 for(j = 0; j < 4; j++){  383 if(block[currentBlock][blockRotate][i][j] == 1)  384 {  385 f[blockY+i][blockX+j] = 1;  386  387  388 }  389 }  390 } |

이 함수는 필드에 block을 추가하는 함수로, block이 null이 아니라면 4\*4의 블록을 그 필드에 위치시키는 함수이다. 이 함수의 시간 복잡도는 4\*4의 이중 for문을 사용했기에 O(16)이어서 O(1)에 해당한다. 공간 복잡도는 마찬가지로 고정된 변수를 사용하기 때문에 O(1)이다.

1-5)

|  |
| --- |
| int DeleteLine(char f[HEIGHT][WIDTH])  // for문을 사용해서 바닥부터 정보를 읽는다.  For  For  필드의 값이 1로 가득찬 줄이 있을 때  이 줄을 없애고 위에 있는 줄의 정보들을 받아서 다시 새로 그린다. |

|  |
| --- |
| int DeleteLine(char f[HEIGHT][WIDTH]){  401 // user code  402  403 //1. 필드를 탐색하여, 꽉 찬 구간이 있는지 탐색한다.  404 //2. 꽉 찬 구간이 있으면 해당 구간을 지운다. 즉, 해당 구간으로 필드값을 한칸씩 내린다.  405 int i, j;  406 int ni, nj;  407 int k;  408 int full\_flag = 0;  409 int numofline = 0;  410  411 for(i=HEIGHT-1; i>=0; i--){  412 for(j=0; j<WIDTH; j++) {  413 if(f[i][j] == 0)  414 break;  415 if(f[i][j] == 1 && j == WIDTH -1)  416 full\_flag = 1;  417 }  418 // 현재 i번째 줄이 공백이 없는 줄이다.  419 if(full\_flag == 1){  420 // i번쨰 줄부터 시작  421 for( ni = i; ni > 0 ; ni--)  422 {  423 for( nj = 0; nj <WIDTH; nj++)  424 {  425 f[ni][nj] = f[ni-1][nj];  426 }  427 }  428 // ni가 1로 갈 때까지밖에 안 함 index가 범위 초  과하기 때문에  429  430 for(k = 0; k <WIDTH; k++)  431 {  432 f[0][k] =0;  433 }  434 full\_flag = 0;  435 i++;  436 numofline += 1;  437 }  438 }  439  440 return ((numofline)\*(numofline)\*100);  441 } |

psedu code를 작성할 때 막연하게 필드를 이중 loop를 이용해서 읽으라고 생각을 하지만, 초기화하는 과정에서 필드 맨 위에서 아래로 움직이며 필드를 읽으며 바꾸는 과정으로 코드를 작성한다면, 필드의 메모리 손실이 무조건 있을 수밖에 없어, 의도했던 필드가 사라지게 된다. 따라서 실습 코드에서 아래에서 위로 움직이며 필드를 바꿔줬다. (line 425), 그렇게 되면 for문의 범위를 떠나게 되어 첫번째 줄 필드도 따로 설정해주어야 한다. (line 430 ~ 433). 주어진 함수는 line 411부터 416까지 HEIGHT\*WIDTH의 이중 loop를 돌아가기 때문에 시간 복잡도는 O(HEIGHT \* WIDTH)이다. 공간 복잡도를 구하기 이전의 필드의 크기(2차원 배열의 크기)는 HEIGHT \* WIDTH이지만, 이는 입력에 따라 확장 되지 않기 때문에 공간복잡도 또한 O(1)이라고 할 수 있다.

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

과제의 각 기능인 그림자 기능을 구현하기 위해

void DrawBlock(), void DrawShadow(), void DrawBlockWithFeatures(),

초기 화면에서 추가된 두번째 block을 그리기 위해

void InitTetris(), void DrawNextBlock(int \* nextBlock), void BlockDown(int sig)

점수를 블록이 맞닿은 면적까지 포함하여 score에 추가하기 위해

int AddBlockToField(), void BlockDown()으로 묶어 설명을 하도록 하겠다.

1. 그림자 기능

|  |
| --- |
| 1. void DrawBlock (int y, int x, int blockID, int blockRotate, char tile)  //기존 drawblock은 parameter인 char tile을 받아 그리기 때문에 그림자는 ‘/’를 그려야 한다.  *이는 기존의 DrawBlock의 시간 복잡도나 공간 복잡도가 달라지지 않는다. 공간 복잡도는 입력 값에 따라 데이터의 크기들이 달라지지 않으니 O(1)이라고 할 수 있고, 시간 복잡도는 기존 void DrawBlock 함수에 의하면, 이중 loop문이 쓰였는데 4\*4 크기이기 때문에 결과적으로 시간 복잡도 또한 O(1)이다.*  2. void DrawShadow(int y, int x, int blockID,int blockRotate)  //이는 현재 블록을 가장 아래로 내렸을 때 더 이상 내려갈 수 없는 위치를 그림자의 위치로 결정하고, ‘/’로 그려야 한다.  //while(checkToMov(y++) //y를 계속 증가하면 checkToMove, 즉 더 이상 아래로 내려갈 수 없을 떄의 위치를 기억한다.  //그 위치에 DrawBlock()를 호출하는데 char tile을 ‘/’로 해서 호출한다.  *while 문에서 checkTomove가 계속 호출되기 때문에 checktomove의 시간 및 공간 복잡도 또한 고려해야 한다. 이전에 알 수 있듯이 checkToMove 함수의 시간 복잡도는 O(WIDTH\*HEIGHT)이다. 이를 최악의 경우 HEIGHT만큼 진행할 수 있으므로 시간 복잡도는 O(HEIGHT \* HEIGHT \* WIDTH)이다. 공간 복잡도는 입력된 변수의 크기가 고정되었기 때문에 O(1)이다.*  3. void DrawBlockWithFeatures(int y, int x, int blockID, int blockRotate)  // 숙제 코드에서 DrawBlock을 대신할 함수로, 더 이상 내려갈 수 없는 Block을 그리는 함수이며, 그림자를 계속해서 그리는 함수이다.  //DrawBLock 호출  // DrawShadow 호출  이 함수의 시간 및 공간 복잡도는 호출되는 두 함수 중 큰 값을 따르게 되는데, DrawBLock 함수와 DrawShadow 함수는 모두 공간 복잡도가 O(1)이므로, DrawBlockWithFeatures의 공간 복잡도는 O(1)이라고 볼 수 있다. 시간 복잡도의 경우에는 DrawBlock은 4\*4으로 이루어진 for\_loop으로 돌기 떄문에 O(1)이지만, DrawShadow는 O(HEIGHT\*WIDHT)이다. 따러서, DrawBlockWIthFeatures 함수의 시간 복잡도는 O(HEIGHT \* WIDTH)이다. |

2. 초기 화면 기능

|  |
| --- |
| 1. void InitTetris()  //여기서 nextblock[2] 부분만 추가하고 DrawBlock 부분을 바꾸면 된다.  *따라서 시간 복잡도와 공간 복잡도는 기존의 InitTetris()와 다르지 않다. 이 함수는 고정된 변수를 사용하기 때문에 시간 복잡도는 O(1), 공간 복잡도는 O(1)이다.*  시간  2. void DrawNextBlock(int \* nextBlock)  //이 함수는 InitTetris에서 추가로 집어넣은 NextBlock[2]부분을 추가해서 게임화면에 나타나는 함수로,  For 블록의 크기 4  For 블록의 크기 4  If(block이 있을 떄)  //그리기  *부분을 추가하면 된다. 이는 상수의 곱으로 이루어진 for loop이므로 시간 복잡도의 크기는 O(1)이다. 공간 복잡도는 물론 함수에서 nextBlock이라는 배열이 쓰이지만, 이것의 크기는 고정되어있기 때문에 공간 복잡도는 O(1)이다.*  3. void BlockDown(int sig)  //blockdown에서 nextblock을 다음 블록으로 초기화하는 과정을 수정하면 된다.  *이는 시간 복잡도와 공간 복잡도에 지장을 주지 않기 떄문에 기존의 시간 복잡도 및 공간 복잡도인 O(1)과 O(1)와 같다.* |

3. 추가 점수 기능

|  |
| --- |
| 1. int AddBlockToField()  //touched라는 변수를 설정한다,  // int touched =0;  // for문 이후 if((f[blocky+i+1][blockX+j]가 1일 때)) 이것은 바로 밑에 블록이 있다는 뜻이다.  그리고 blocky+i가 필드의 최대 높이까지 갔을 때를 확인해서 touched 변수를 증가시킨다.  *이는 기존 for loop에 조건문을 추가했기 때문에 시간 복잡도와 공간 복잡도는 달라지지 않고, 시간 복잡도는 O(1)이며, 공간 복잡도는 O(1)에 해당한다.*  2. void BlockDown()  // AddBlockToField()를 수정했다. return 값도 정수형으로 바꿨기 떄문에 score를 수정하는 구문을 바꾸면 된다.  // AddBlockToField() -> 호출하면 score값을 return 하며 초기화 된다.  // score += DeleteLine()  *시간 및 공간 복잡도에 관한 이야기는 위 BlockDown() 함수와 같기 때문에 생략하도록 하겠다.* |