테트리스 1주차 결과보고서

전공: 철학과 학년: 3학년 학번: 20180032 이름: 남기동

1. 예비 보고서의 pseudo code와 실습의 차이 & 실습 코드의 시간 공간 복잡도

1) CheckToMove 함수

예비 보고서의 pseudo code와 크게 다른 점은 없었지만 i와 j가 HEIGHT와 WIDTH만큼 움직이는 것이 아니라 블록의 사이즈를 나타내는 4 x 4 행렬에 맞게 0부터 4까지 변화하는 것이 달랐다. 블록의 4 x 4 행렬 중에서 공간이 차 있는 곳의 좌표를 구해서 기존 blockY, blockX에다가 더한 필드의 공간이 차 있는지 판단하는 것이다.

i, j의 이중 루프를 통해 currentBlock에서 차 있는 공간을 파악하여 필드 바깥으로 나가거나 필드와 충돌하는 경우를 체크하고 그렇지 않으면 1을 리턴하는 구조를 갖는다. 각각의 블록이 n x n 행렬에 표현되는 구조라면 시간 복잡도는 O(n^2)이다. 공간 복잡도는 i와 j를 저장하기 위한 변수는 고정된 크기를 갖고 있으며 인자로 받는 필드가 HEIGHT x WIDTH 만큼의 2차원 배열이고 블록의 정보도 저장해야 하기 때문에 공간 복잡도는 O(height x width + n^2)이다

2) DrawChange 함수

예비 보고서에서 Drawchange에 대해서는 간략하게 pseudo code를 짰기 때문에 내용적으로 크게 다른 점은 없지만 실질적으로 DrawChange 함수를 구현하기 위해서는 블록의 이전 정보를 구해서 기존 블록의 정보를 지우는 작업에 필요한 요소가 복잡했다. KEY_UP의 경우에는 회전이 달라지는 것이기에 이전 블록의 회전 정보를 구하기 위해 pre_Rotate = (blockRotate + 3) % 4를 사용했다. KEY_DOWN의 경우에는 y 좌표에 변화를, KEY_LEFT, KEY_RIGHT는 x 좌표에 변화를 주어서 이전 블록의 위치에 '.'을 넣어서 지우고 DrawBlock을 통해 새로운 위치의 블록을 그리게 된다.

시간 복잡도의 경우에는 블록이 n x n 행렬이라고 한다면 이전 블록의 위치를 지우기 위해 i와 j

의 이중 루프를 사용하여 n x n 행렬을 탐색해야 한다. 이러한 탐색이 서로 다른 4가지 키보드 입력에 대해 각각 존재하고 마지막 DrawBlock 함수를 호출하기도 하지만 이 역시 i와 j의 이중 루프로 n x n 행렬의 블록 정보를 탐색하는 것은 동일하다. 따라서 시간 복잡도는 O(n^2)이다. 공간 복잡도는 인자로 받는 필드의 정보를 저장해야 하고 블록의 정보도 저장해야 하기 때문에 공간 복잡도는 O(height x width + n^2)

3) BlockDown 함수

예비 보고서와 다른 점은 CheckToMove 함수로 내려갈 수 있는지 검사하고 난 이후에 내리기 전에 이전 위치의 블록을 지워야 한다는 점이 추가되었다는 것이다. 예비 보고서에서 pseudo code를 작성할 때는 이러한 부분에 대해 간과하고 코드를 설계했다. 그 외에 만약 1칸을 내릴 수 없다면 addblocktofield함수, deleteline함수 등을 호출하고 이후 작업을 위한 처리를 하는 것들은 예비 보고서의 pseudo code와 동일했다.

blockdown함수는 1칸 내릴 수 있는 경우와 내릴 수 없는 경우로 구분되는데, 전자의 경우에는 checktoMove함수나 혹은 이전 블록을 지우는 것이나 drawblock함수 모두 O(n^2)의 시간 복잡도를 갖고 있다. 후자의 경우에는 addblocktofield함수나 deleteline함수나 drawnextblock함수 모두 O(n^2)의 시간 복잡도를 갖고 있기 때문에 BlockDown 함수의 시간 복잡도는 O(n^2)이다.

공간 복잡도는 역시 필드의 값을 저장해야 하고 블록의 정보도 저장해야 하기 때문에 공간 복잡도는 O(height x width + n^2)

4) AddBlockToField 함수

AddBlockToField 함수도 pseudo code를 작성할 때 ppt 자료를 참고하여 i와 j가 height와 width 만큼 변화한다고 작성하였지만 실질적으로 실습에서 작성해보니 i와 j는 블록의 4 x 4 행렬을 탐색하는 용도로 사용되기 때문에 0부터 4까지 변하게 된다.

i, j의 이중 루프를 통해 currentBlock을 조사하고 차있는 공간에 대응하는 필드의 위치를 1로 대입하는 구조를 갖는다. 각각의 블록이 n x n 행렬에 표현되는 구조라면 시간 복잡도는 O(n^2)이다.

공간 복잡도는 i와 j를 저장하기 위한 변수는 고정된 크기를 갖고 있으며 가장 큰 공간을 차지하는 것은 인자로 받은 필드의 정보를 저장하고 블록의 정보도 저장해야 하기 때문에 공간 복잡도는 O(height x width + n^2)이다.

5) DeleteLine 함수

DeleteLine 함수의 경우에는 예비 보고서에서 설계한 pseudo code와 흡사하지만 꽉찬 줄이 있어서 이를 삭제한 이후 그 위의 필드들을 한 칸씩 아래로 이동시키는 작업에서 k라는 변수를 하나 더 설정하여 for (k=i, k >=0; k--) 이러한 for 문을 작성해야 제대로 작동하였다. 이외에는 예비 보고서의 pseudo code와 실습의 내용이 동일했다.

i와 j의 이중 루프를 통해 필드 값을 모두 조사하면서 꽉 찬 줄을 발견하면 해당 줄을 삭제하고 그 위의 필드를 한 칸씩 아래로 이동시키는 작업을 수행한다. Worst case를 따져보면 모두 꽉찬 줄이어서 위에서부터 한 줄씩 지우고 내리기를 모든 줄마다 반복해야 하기 때문에 HEIGHTxWIDTHxHEIGHT번 반복해야 한다. 따라서 HEIGHT를 h로 WIDTH를 w로 가정한다면, 시간 복잡도는 O(w*h^2)이다. 공간 복잡도는 i, j, k, count, deleteflag 모두 고정된 크기를 갖고 있으며 가장 큰 공간을 차지하는 것은 인자로 받은 필드의 정보를 저장하고 블록의 정보도 저장해야 하기 때문에 공간 복잡도는 O(height x width + n^2)이다.

2. 과제를 해결하기 위한 pseudo code와 시간, 공간 복잡도

1) DrawShadow 함수 → 1번 그림자 기능을 위해 생성

```
tmp_blockY = blockY
While( CheckToMove != 0 )
        tmp_blockY++
DrawBlock(tmp_blockY, blockX, blockID, blockRotate, '/')
```

필드의 HEIGHT가 h라고 가정하고 worst case를 생각하면, 현재 블록은 필드의 맨 위에 위치하고 필드는 맨 아래일 때이고 이 경우 while문은 h만큼 반복될 것이다. 그리고 n x n 행렬의 블록을 그린다고 했을 때 drawblock 함수는 O(n^2)의 시간 복잡도를 갖는다. 따라서 DrawShadow함수는 O(n^2 + h)의 시간 복잡도를 갖는다. 이 함수에서 호출하는 checktoMove함수가 필드의 영역을 값을 갖고 있어야 하고 동시에 블록의 정보도 저장해야 하기 때문에 공간 복잡도는 O(height x width + n^2)이다.

2) DrawBlockWithFeatures 함수 → 1번 그림자 기능을 위해 생성

DrawShadow

DrawBlock

굳이 이 함수를 구현하지 않고 기존에 drawblock이 쓰인 곳에서 drawshadow함수를 먼저 호출하는 방향으로 하면 되어서 실질적으로 이 함수를 만들지는 않았다. 하지만 이 함수를 만든다면 drawshadow함수와 drawblock함수를 연달아 호출하면 될 것이고, drawshadow나 drawblock 모두 블록이 n x n 행렬이라고 하였을 때 O(n^2)의 시간 복잡도를 갖는다. 따라서 이 함수의 시간 복잡도도 O(n^2)이다. 공간 복잡도는 drawshadow함수가 호출하는 checktoMove함수가 필드의 영역을 값을 갖고 있어야 하고 동시에 블록의 정보도 저장해야 하기 때문에 공간 복잡도는 O(height x width + n^2)이다.

3) DrawNextBlock 함수 → 2번 기능을 위해 수정

int i, j

for i = 0 to 4 //첫번째 nextblock 그리기

    다음 블록 그리라고 만든 테두리로 move

    for j=0 to 4

    if(nextBlock[1]이 채워진 곳이라면)

        빈공간을 색 반전하여 채워진 타일 그리기

Else 빈공간을 공백으로 표시

for i = 0 to 4 //두번째 nextblock 그리기

다음 블록 그리라고 만든 테두리로 move

for j=0 to 4

if(nextBlock[2]가 채워진 곳이라면)

빈공간을 색 반전하여 채워진 타일 그리기

Else 빈공간을 공백으로 표시

기존에 하나의 다른 nextblock을 그리던 것에서 두개의 nextblock을 그릴 수 있도록 수정해야 했다. 크게 다른 부분은 없고 첫 nextblock을 그리는 코드 그대로 두번째 nextblock을 그리게 사용하면 된다. 우선 nextblock을 그리는 것은 블록이 n x n행렬일 때 nextblock[1]의 n x n 행렬을 조사하기 때문에 O(n^2)의 시간 복잡도를 갖는다. 이것이 두 번 반복된다고 해도 시간 복잡도는 빅오 노테이션에서 상수가 고려되지 않기 때문에 동일하다. 따라서 수정된 함수의 시간 복잡도도 여전히 O(n^2)이다. 공간 복잡도는 블록이 n x n 행렬일 때 해당 블록의 행렬에 값의 정보를 갖고 있어야 하기 때문에 O(n^2)이다.

4) AddBlockToField 함수 → 3번 score 기능을 위해 수정

int i, j, touched

for I = 0 to 4

for j = 0 to 4

If 현재 블록 == 1(차있다면)

해당 필드의 값을 1로 변경

If(해당 필드의 한칸 아래가 1이라면) touched++

Else if(한칸 아래가 필드의 맨바닥이면) touched++

return touched*10

이 함수에서 달라진 점은 touched 변수를 통해 해당 필드의 한 칸 아래가 1이거나 한 칸 아래가 필드의 맨 바닥일 때 touched의 값을 증가시키는 것뿐이다. 따라서 시간 복잡도는 여전히 블록이 n x n 행렬일 때 O(n^2)일 것이며 공간 복잡도는 블록의 정보를 저장해야 하고 필드의 정보도 저장해야 하기 때문에 O(height x width + n^2)이다.