8주차 결과보고서

전공 : 경영학과 학년 : 4학년 학번 : 20190808 이름 : 방지혁

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

int CheckToMove: 블록이 움직일 수 있는지 확인

기존의 Pseudo 코드에서는, 아래, 왼쪽, 오른쪽에 대해 개별적으로 경계확인을 했었습니다. 그러나 이를 모두 한 줄로 합쳤습니다. if (i + blockY >= HEIGHT || j + blockX < 0 || j + blockX >= WIDTH || f[i + blockY][j + blockX] == 1) 조건 체크를 다음과 같이 OR를 사용하여 작성해서 더 간결하게 만들었습니다. 시간 복잡도의 경우 가로 및 세로 4칸 배열에 대해 for 루프가 돌아가면서 검사하는 것이기에 상수시간으로 O(1)이라고 볼 수 있습니다. 공간 복잡도의 경우 추가적으로 for loop 반복을 위한 변수 외에는 추가적으로 공간을 할당하지 않았기 때문에 O(1)입니다.

void DrawChange: 예전 블록을 지우고 다시 그려줌

```
// 이전 블록이 그려진 거 지우기
        FOR i = 0 to 3:
                FOR j = 0 to 3:
                IF block[현재블록][이전회전][i][j] == 1 AND i +  이전Y >= 0:
                        커서 이동
                        "." 출력
```

기존에는 drawblock 함수를 사용하여 블록을 지우려고 했지만 역상 이슈가 있어서 역상시키는 부분만 제외하고 drawblock의 그리는 함수만 따로 가져와 구현했으며 이 과정에서 예비 보고서에 썼던 기존 pseudo 코드 DrawBlock(prevBlockY, prevBlockX, currentBlock, prevBlockRotate, '.') 도 수정했었습니다. 시간 복잡도의 경우 4X4 크기의 배열을 순회하므로 O(1)이고, 공간 복잡도의 경우 이전과 같이 O(1)입니다.

void BlockDown(int sig): 매초마다 블록을 한 칸씩 내림

기존 pseudo code에서는 blockY == -1이면 게임 오버로 처리하는 코드를 새 블록 배치 후에 위치했었지만, 이를 이렇게 바로 체크하도록 구현했습니다.

```
if (blockY == -1) gameOver = 1;
AddBlockToField(field, nextBlock[0], blockRotate, blockY, blockX);
score += DeleteLine(field)
```

해당 함수에서는 CheckToMove, AddBlockToField, DeleteLine을 호출합니다. CheckToMove, AddBlockToField 모두 4 X 4 배열을 순회만 하기에 시간복잡도는 O(1)입니다. 다만 deleteLine은 후술하겠지만 O(200)으로 마찬가지로 O(1) 즉 상수 시간 복잡도를 가지고 있습니다. 굳이 따지자면 O(Height * Width)의 시간복잡도를 가지고 있다고 볼 수 있습니다. 공간복잡도의 경우 새로 공간을 할당하거나 그렇지 않기 때문에 O(1)이라고 볼 수 있습니다.

void AddBlockToField: 블록을 필드에 쌓음

pseudo 코드와 같으며 이전의 함수들과 마찬가지로 4X4배열을 돌며 조건 체크만 하는 단순한 함수이기에 시간복잡도는 O(1), 공간복잡도는 O(1)이라고 볼 수 있습니다.

int DeleteLine: 채워진 가로줄을 삭제

pseudo 코드에서는 while문을 사용했었습니다.

```
i = HEIGHT – 1 // 맨 아랫줄에서 시작해야해서
        WHILE i >= 0:
                // 현재 줄이 꽉 찼는지 확인
                j = 0
                FOR j = 0 to WIDTH – 1:
                        If field[i][j]가 비어있으면:
                                Break // 빈 칸 발견, 이 줄은 꽉 안 참
                // 어? 현재 줄이 꽉 차 있네
                IF j == WIDTH:
                        삭제된_줄_개수++

                        FOR k = i to 0:
                                IF k == 0: // 맨 윗 줄인 경우
                                        k번째 줄을 0으로 초기화
                                ELSE:
                                        k번째 줄 = k-1번째 줄
                        i는 그대로 유지
                ELSE:
                        // 꽉 안 찼음
                        i—
```

그러나 실제로 구현할 때는 for 문으로 처리했습니다. 대신 줄을 삭제하고 내려올 때, 같은 인덱스에 대해서 다시 검사해야하기 때문에 인덱스 감소 부분은 조건부로 되도록 구현했습니다.

```
for (int i = HEIGHT - 1; i >= 0;) {
      int j = 0;
      for (; j < WIDTH; j++) {
        if (f[i][j] == 0) break;
      }
      if (j == WIDTH) {
        // 해당 라인이 꽉 찼음
        deleted_lines_cnt++;
        // 위의 라인들 지우기
        for (int k = i; k >= 0; k--) {
          if (k == 0) {
            // 최상단 라인인 경우 0으로 초기화
            for (int l = 0; l < WIDTH; l++) {
              f[k][l] = 0;
            }
          }
          else {
            for (int l = 0; l < WIDTH; l++) {
              f[k][l] = f[k - 1][l];
            }
          }
        }
      }
      else {
        i--;
      }
    }
```

변수 선언 복잡성을 낮추기 위해 해당 구현으로 바꾸었고 로직은 비슷합니다. 시간 복잡도의 경우 O(HEIGHT * WIDTH)입니다. 최악의 경우 모든 줄이 꽉 차서 HEIGHT만큼 반복하고, 각 iteration마다 WIDTH만큼 검사해야 하기 때문입니다. 그러나 상수 복잡도이기에 O(1)이라고 볼 수 있습니다. 공간 복잡도의 경우 반복문을 위한 iteration변수 외에는 따로 공간을 할당 및 사용하지 않았기 때문에 O(1)입니다.

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

그림자 기능

DrawShadow 및 Draw BlockWith Features의 pseudo code

```
함수 DrawShadow(y ,x, blockID, blockRotate):
        shadowY = y + 1 // 현재 위치 바로 아래에서 시작할 수 있도록

        // 더 이상 내려가지 못할 때까지 반복
        WHILE CheckToMove(filed, blockID, blockRotate, shadowy, x) == 1
                shadowy += 1

        // 마지막으로 이동 가능했던 부분으로 다시 가기
        shadowY—

        // 그림자 그리기
        DrawBlock(shadowY, x, blockID, blockRotate, '/')

        END

함수 DrawBlockWithFeatures(y, x, blockID, blockRotate):
        // 현재 블록 먼저 그리기
        DrawBlock(...)
        / / 그림자 그리기
        DrawShadow(...)
        END
```

시간 복잡도의 경우 loop가 최대 HEIGHT번 즉 O(20)이며 각 반복마다 checkToMove를 호출하는데 checkToMove 같은 경우 시간복잡도가 O(1)이고 또 마지막으로 O(1)의 drawblock을 호출하기에 결국 시간복잡도는 이전과 동일하게 O(1)이라고 볼 수 있습니다. 그리고 drawchange도 수정해줍니다. Drawshadow와 drawblockwithfeatures 함수 둘 다 별도의 변수 선언이 없기에 공간복잡도는 O(1)입니다.

Drawchange 수정

```
함수 DrawChange(...):
        // 이전 블록 상태 계산
        prevBlockRotate = blockRotate
        prevBlocky = blockY
        prevBlockx = blockX

        switch(command):
        생략...

        // 이전 블록 지우기
        For int i = 0 to 3:
                For int j = 0 to 3:
                        만약 block[currentBlock][prevBlockRotate][i][j] == 1이고
                        범위에서 벗어나지 않는다면
                        Printw(".")

        // 이전 그림자 위치 계산
        Int shadowy = prevBlockY + 1
        Int shadow = prevBlockX

        While(CheckToMove(...)) shadowy++

        Shadowy—

        // 이전 그림자 지우기
        For int i = 0 to 3:
                For int j = 0 to 3:
                        만약 block[currentBlock][prevBlockRotate][i][j] == 1이고
                        범위에서 벗어나지 않는다면
                        Printw(".")

        DrawBlock(...)
        / / 그림자 그리기
        DrawShadow(...)
        END
```

시간과 공간 복잡도는 상수시간 복잡도로 이전과 같습니다.

2개의 블록 미리 보여주기

InitTetris함수 & DrawNextBlock함수 & BlockDown 함수

```
함수 InitTetris(...):
        // 기존 초기화 코드

        // 다음 블록 3개 생성

        nextBlock[0] = rand() % 7
        nextBlock[1] = rand() % 7
        nextBlock[2] = rand() % 7



        // 나머지 초기화
        RETURN

함수 DrawNextBlock(...):
        // 1. 첫 번째 다음 블록 그리기
        For int i = 0 to 3:
                For int j = 0 to 3:
                        IF block[nextBlock[1]][0][i][j] == 1:
                                반전된 공백 출력
                        ELSE:
                                일반 공백 출력
        // 2. 두 번째 다음 블록 그리기
        For int i = 0 to 3:
                For int j = 0 to 3:
                        IF block[nextBlock[1]][0][i][j] == 1:
                                반전된 공백 출력
                        ELSE:
                                일반 공백 출력
        RETURN

함수 DrawDown(...) 수정:
        // 블록을 더 이상 내릴 수 없을 때
        // 기존 코드

        nextBlock[0] = nextBlock[1]
        nextBlock[1] = nextBlock[2]
        nextBlock[2] = rand() % 7
        DrawNextBlock()
```

InitTetris는 단순히 변수에 대입만 추가했기에 시간 복잡도는 O(1), DrawNextBlock은 4 X 4 블록 1개 더 그리는 것을 추가했기에 마찬가지로 시간복잡도는 O(1), 공간 복잡도도 O(1)입니다. BlockDown의 경우도 대입 연산만 추가했기에 시간복잡도는 O(1)입니다. InitTetris와 BlockDown 모두 nextBlock 배열 크기만 1이 증가 했기에 공간 복잡도는 기존처럼 O(1)입니다.

닿은 면적만큼 score 증가하기

AddblockToField 함수 및 BlockDown 함수 수정

```
함수 AddBlockToField(...):
        Touched_bottom = 0   // 접촉 면적 카운터
        // 접촉 면적 계산
        FOR i = 0 to 3:
                FOR j = 0 to 3:
                // 블록의 해당 위치에 실제 블록이 있으면
                        IF block[currentBlock][blockRotate][i][j] == 1:
                                // 바닥에 닿음
                                IF i + blockY + 1 == HEIGHT - 1:
                                        Touched_bottom++
                                // 다른 블록 위에 놓임
                                ELSE IF field[i + blockY + 1][j + blockX] == 1:
                                        Touched_bottom++
                                // 필드에 블록 추가
                                field[i + blockY][j + blockX] = 1
        RETURN touch_bottom * 10

함수 BlockDown(...) 수정:
        ...
        IF CheckToMove(...) == 0:
                blockY++
                DrawChange()
        ELSE:
        // 블록을 더 이상 내릴 수 없을 때
        // 1. 블록을 필드에 추가하고 접촉 점수 받기
                touchScore = AddBlockToField(...)
        // 2. 삭제된 라인 점수 계산
                deletedScore = DeleteLine(...)
        // 3. 총 점수 누적
                score += touchScore + deletedScore
       // 나머지 코드...
```

```
        PrintScore(score)
    RETURN
```

기존에는 아무것도 return하지 않던 addblocktofield함수에 대해 block이 맨 아랫줄에 닿았거나 아래에 블록이 있을 경우 해당 접촉 면적들을 세어서 BlockDown 함수에서 기존 점수에 더해줍니다. 마찬가지로 시간 및 공간 복잡도는 addblocktofield는 4 X 4 배열만 순회하기에 O(1)이고 blockdown도 O(1)입니다. 또한 카운터 변수 하나만 더 추가되었기에 공간 복잡도도 둘 다 O(1)입니다.