10주차 결과보고서

전공: 기계공학과 학년: 3학년 학번: 20191820 이름: 김형준

**1.**

1-1. recommended() 함수 :

typedef struct \_RecNode{

int level; // 탐색 레벨을 저장함 (헤드 노드=0, 자식노드로 갈수록 1씩 올라감)

int accumulatedScore; // 저장된 점수값

char recField[HEIGHT][WIDTH]; // 지정된 위치에 블록을 쌓았을 때의 테트리스 필드 데이터를 저장함

int passflag; // 점수의 최대값 계산시 제외할지의 여부, 1이면 스킵함

struct \_RecNode \*\*child; // 자식 노드들의 주소를 저장하는 배열의 포인터

int childsize; // 자식노드의 총 개수 (항상 40으로 설정함)

int curBlockID; // 놓을 블록의 종류(ID)

int recBlockX,recBlockY,recBlockRotate; // 각각 블록을 놓을 X, Y좌표, 블록의 회전상태

struct \_RecNode \*parent; // 부모 노드의 주소

} RecNode;

RecNode \*recRoot;

추천 기능을 구현하는데 사용한 자료구조는 트리이며, 위와 같은 구조체(RecNode)로 구현되었다. 전역변수 recRoot에 트리의 헤드 노드의 주소를 저장하고 recommended() 함수의 인자로 전달해 추천 블록 탐색 기능을 실행한다.

InitTetris() 함수에서 xStart의 값을 초기화하고 헤드노드를 할당한 다음 값을 초기화 한다.

그런 다음, recommend(recRoot)를 호출해 게임 시작시 첫 블록의 추천위치를 계산한다.

두 번째 블럭부터는 BlockDown() 함수에서 헤드 노드의 값을 초기화하고

recommend(recRoot)를 호출해 블록이 바뀔 때 마다 추천위치를 다시 그리게 한다.

(xStart는 각 블록ID와 그 블록의 회전상태에 대해서 해당 블록이 가질 수 있는 X좌표의 최소값을 저장하는 배열임, 블록의 모양은

block[NUM\_OF\_SHAPE][NUM\_OF\_ROTATE][BLOCK\_HEIGHT][BLOCK\_WIDTH]에서

각각의 모양이 4x4 박스로 정의되어있는데, 박스의 맨 왼쪽 위의 칸을 블록위치로 계산하므로 4x4 박스의 맨 왼쪽줄에 블록이 모두 채워져 있지 않다면, X좌표가 0보다 더 작은 값을 가질 수 있다. 따라서 xStart에서는 실제로 블록이 있는 칸이 테트리스 필드 왼쪽 벽에 닿을 때의 X좌표값을 저장한다.)

recommended() 함수에서는 호출된 노드를 2중 루프로 반복시켜 현재 놓을 블록(root-> curBlockID)의 모든 경우의 수를 검사한다.

현재 노드의 자식노드가 아직 할당이 되지 않은 경우에는 자식노드를 할당하고, 자식노드의 level이 VISIBLE\_BLOCKS 미만일때 자식노드의 자식노드도 함께 할당한다. VISIBLE\_BLOCKS 이상이면 자식노드의 level이 VISIBLE\_BLOCKS에 도달했으므로 leaf node로 만든다. (자식노드를 추가로 생성하지 않음)

자식 노드의 passflag를 0으로 초기화하고 nextBlock에서 다음에 놓을 블록의 ID를 curBlockID에 저장한다. 그런 다음 현재 노드에서 memcpy로 (2중 배열을 사용하는 것 보다 속도가 빠르므로 대신 사용함) 자식 노드로 recField를 복사한다.

그 다음, 현재 경우(2중 루프문에서의 i(회전상태), j(X좌표 순서, 맨 왼쪽부터 0))에서 블록을 놓았을 때의 위치를 계산한다. (curRotate : 회전상태, curBlockX : X좌표, curBlockY : Y좌표, 이때 curBlockY가 -1이면 해당 경우에서는 블록을 놓을 수 없는 상태이므로 바로 다음 경우로 넘어감)

위에서 계산한 위치 정보를 사용해 블록을 놓았을 때 얻는 점수를 자식노드에 계산한다.

(이때 recField의 정보도 계산한 위치에 블록을 추가했을 때의 상황으로 갱신됨)

자식 노드의 level이 VISIBLE\_BLOCKS 미만이면 자식노드로 재귀호출하여 현재 상태에서 VISIBLE\_BLOCKS까지의 모든 블록을 놓았을 때의 최대 점수값을 curScore에 저장한다.

VISIBLE\_BLOCKS 이상이면 자식노드의 accumulatedScore를 리턴한다.

이때, 자식노드의 passflag가 1인 경우 다음 블록을 모든 경우에서 놓을 수 없다는 뜻이므로 최대값 계산에 고려하지 않고 다음으로 넘어간다.

curScore가 max보다 크면 max를 갱신하고 현재 노드의 recBlockX, recBlockY,

recBlockRotate를 현재 블록을 놓은 위치와 회전상태로 갱신하여 추천할 블록의 위치 정보를 저장한다. (DrawRecommend() 함수에서 recRoot의 recBlockX, recBlockY, recBlockRotate를 사용하여 추천위치를 그리게 된다.)

마지막으로, check가 0이면 현재 노드의 passflag를 1로 설정해 최대값 계산에서 제외되게 설정한다. (max값 갱신시 check에 1을 저장하므로 2중 루프문을 돌고 나서도 check가 0이면 모든 자식노드에서 현재 블록을 놓을 수 없어 추천 위치를 결정하지 못한 상태이므로 이 경우, 현재 노드는 최대값 위치 계산에서 제외되어야 한다.)

recommended() 함수는 VISIBLE\_BLOCKS까지만 재귀호출하고, 내부에서 시간복잡도가 가장 큰 부분은 memcpy를 사용해 recField를 복사하는 부분이다. (recField의 크기는 O(WIDTH\*HEIGHT)이고, 모든 자식노드에 대해서 각각 복사하므로 시간복잡도는

O(WIDTH^2\*HEIGHT\*40^(VISIBLE\_BLOCKS)) 이다.

각 노드는 공간복잡도가 O(WIDTH\*HEIGHT)인 recField를 가지고, leaf node를 제외하면 모든 노드의 childsize는 40이므로 공간복잡도는

O(WIDTH\*HEIGHT\*40^(VISIBLE\_BLOCKS)) 이다.

1-2. modified\_recommended() 함수 :

modified\_recommended() 함수에서는 recommended()에서 pruning tree을 사용하고,

pruning 할 때의 점수와 최종 점수를 분리하여 고득점을 할 수 있도록 변경하였다.

typedef struct \_RecNode{ // 나머지 항목들은 모두 이전과 동일함

int level;

int key; // pruning시 순위를 결정할 점수

int accumulatedScore;

char recField[HEIGHT][WIDTH];

int passflag;

struct \_RecNode \*\*child;

int childsize;

int curBlockID;

int recBlockX,recBlockY,recBlockRotate;

int curBlockX,curBlockY,curBlockRotate; // 현재 좌표 및 회전상태를 임시로 저장하는 멤버 변수들

struct \_RecNode \*parent;

struct \_RecNode\* RankList[40]; // 자식 노드들의 순위를 저장할 배열

} RecNode;

InitTetris()에서 yLoc을 초기화 하는 부분이 추가되었다.

yLoc은 블록이 놓였을 때, 실제 블록이 있는 위치의 y값과 4x4 박스의 맨 왼쪽 위의 점과의 y좌표의 편차 값을 저장하는 배열이다. (후에 블록이 쌓인 후 전체 블록의 높이를 계산할 때 사용됨)

modified\_recommended() 함수에서는 먼저 FindHeight()를 호출해 현재 노드에 쌓여있는 블록의 최대 높이를 계산한다. 그 다음 상위 16개의 노드를 남기는 부분(pruning)과 남은 노드들 중 점수의 최댓값으로 추천 위치를 결정하는 부분이 순서대로 진행되는데, pruning하는 부분에서 먼저 recommended()에서와 같이 자식노드가 할당이 안되어 있으면 공간 할당 및 초기화를 진행한다. 그 다음 현재 경우에 대해 블록을 놓았을 때의 위치를 계산한다. 계산 후 curBlockY이 -1이면 실제로 가능한 경우가 아니므로 다음 노드로 넘기고, 현재 노드의 recField를 자식노드에게 복사하고 점수(accumulatedScore)와 key값을 계산한다. 점수값 계산시 modified\_AddBlockToField를 사용하는데, 이 함수는

AddBlockToField에서 side\_touched 항목을 추가한 점수이다. (블록의 옆면이 벽이나 다른 블록에 닿을 경우 닿은 면적당 20점으로 계산됨, 좌/우 별도로 계산)

따라서 점수는 줄을 삭제하는데 얻는 점수(DL, 100\*Line^2) + 블록의 바닥에 닿은 점수(AB, touched\*10) + 블록의 옆면에 닿은 점수(AB, side\_touched\*20)로 계산된다.

key는 블록이 놓이는 Y좌표 점수(curBlockY) + 블록을 놓은 후 테트리스 필드에 쌓인 블록들의 높이 점수(HEIGHT-CurH) + 줄을 삭제하는데 얻는 점수(DL, 100\*Line^2)로 계산된다. 이렇게 모든 경우에서 계산이 완료 되면, Merge sort를 사용해 key값으로 RankList를

내림차순으로 정렬해 상위 16개(=SelNum)만을 계속 진행시킨다.

상위 16개의 노드들은 recommended()와 마찬가지로 curScore를 재귀호출로 계산하고, max값을 갱신하여 최종적으로 현재 블록을 추천할 위치를 현재 노드에 저장하게 된다.

recommended()와 마찬가지로 recField를 복사하는 부분의 시간복잡도가 가장 크고,

첫 호출에서는 40개의 자식노드가 생성되고 나머지 재귀호출에서는 pruning된 16개의 노드만이 자식노드를 생성하므로 시간복잡도는

O(WIDTH^2\*HEIGHT\*16^(VISIBLE\_BLOCKS)) 이다.

각 노드는 크기가 O(WIDTH\*HEIGHT)인 recField를 가지고, 첫 호출에서는 40개의 노드가, 나머지 재귀호출에서는 pruning된 16개의 노드만이 자식노드를 생성하므로 공간복잡도는 O(WIDTH\*HEIGHT\*16^(VISIBLE\_BLOCKS)) 이다.

**2.**

▪ pruning tree를 사용하여 시간복잡도 및 공간복잡도의 지수 항의 밑을 40에서 16으로 크게 줄여 프로그램의 성능을 개선시켰다. (자식노드 40개중 상위 16개의 key값을 가지는 노드만을 계속 진행시킴)

▪ passflag를 사용하여 자식노드 전체가 다음 블록을 놓을 수 없는 경우 (=GameOver 상황) 해당 부모노드를 스킵시켜 연산의 효율성을 증가시켰다.

▪ pruning tree로 남길 자식노드를 선택할 때, accumulatedScore 대신 별도로 계산된 key값을 기준으로 대신 사용하여 블록을 추천할 때 가능한 ‘줄을 지워서 점수를 얻는 경우’와 ‘테트리스 전체 높이를 낮게 유지하게 만드는 경우’를 추천하게끔 유도하였다.

▪ accumulatedScore의 계산방식에 side\_touched 항목을 추가하여 줄을 지울 수 없을 때 블록이 위에 쌓이는 것이 아닌, 옆 블록과 밀착시켜 붙이거나 구멍에 들어가게끔 블록 위치를 추천하게 유도하였다.

**3.**

이번 실습과 과제를 진행하면서 pruning tree의 정의와 구현방법에 대해서 자세하게 알게 되었고, 이 자료구조가 지수 시간 알고리즘을 최적화 하는데 유용한 도구로 쓰일 수 있음을 몸소 체험 할 수 있었다. (recommended\_play로 pruning tree를 적용하지 않았을 때는 VISIBLE\_BLOCKS가 3이 사실상 사용할 수 있는 최대 수치였으나, 적용 후에는 4를 넣어도 준수한 속도를 보여주는 것을 알 수 있었다.)

또한, 같은 추천 알고리즘을 구현하더라도 어떻게 테트리스 시스템에 잘 맞춰서 보정(고득점 등)하느냐에 따라 추천 시스템의 성능(진행할 수 있는 최대 점수)가 천차만별로 달라질 수 있다는 것을 알게 되었다. (실습에서 작성한 recommend()는 평균 2만점 정도가 한계지만, 갖가지 최적화와 보정이 들어간 modified\_recommend()는 200만점이 넘어도 잘 끝나지 않았다.)

3주차 코드(실습 및 과제)를 작성하면서 segmentation fault나 예상치 못한 오류들이 많이 떠서 생각한대로 구현하는데 애를 먹었으나, 컴실1 2주차에 배웠던 gdb기능을 활용하여 모두 고칠 수 있었다. 이번 프로젝트를 통해 gdb를 한층 더 능숙하게 다룰 수 있게 된 것 같다.