8주차 결과보고서

전공: 컴퓨터공학과 학년: 2학년 학번: 20191619 이름: 이동석

**1. 실습**

추천 모드 구현을 하기 위해 사용한 자료구조는 다음과 같다.

typedef struct \_RecNode {

float level, accumulatedscore;

int X, Y, R;

char(\*f)[WIDTH];

struct \_RecNode\* c[CHILDREN\_MAX];

} RecNode;

void Init\_recNode(RecNode \* recRoot)

{

RecNode\*\* child = recRoot->c;

for (int i = 0; i < CHILDREN\_MAX; i++) {

\*(child + i) = malloc(sizeof(RecNode));

child[i]->level = recRoot->level + 1;

child[i]->f = (char(\*)[WIDTH])malloc(sizeof(char) \* HEIGHT \* WIDTH);

if (child[i]->level < VISIBLE\_BLOCKS)

Init\_recNode(child[i]);

}

}

우선, 경우의 수를 탐색하기 위해서 블록 각각의 4가지 회전에 대해 모든 x좌표를 확인한다. 각 상황에 대해 움직일 수 있는지 확인하고, 블록을 내릴 수 있는 가장 아래까지 내린다. AddToBlock 함수와 DeleteLine 함수를 이용해 점수를 계산한다. 다음, 아직 depth가 VISIBLE\_BLOCKS 보다 작다면 재귀를 통해 함수를 다시 반복한다. 즉 가장 왼쪽 노드부터 탐색하고 누적된 점수를 저장한다. 어찌보면 DFS 알고리즘과 유사하게 동작한다.

이후 저장된 점수들을 비교하여, 가장 큰 점수를 갖을 때의 x좌표 y좌표 회전을 나타내는 rotate를 전역변수 recommendX,recommendY,recommendR에 저장한다. 넘겨받은 값을 이용해, DrawRecommend함수를 이용해 필드에 그려준다.

이때, 시간복잡도는 O((rotate\*WIDTH)^(VISIBLE\_BLOCKS)) 이 된다. 공간복잡도는 위의 Init\_recNode 함수에서 CHILDREN\_MAX 만큼 자식노드를 생성하고 이를 VISIBLE\_BLOCKS 수만큼 반복하므로, O((CHILDREN\_MAX)^(VISBLE\_BLOCKS)) 가 됨을 알 수 있다.

위 알고리즘의 속도를 향상시키기 위해 pruning 기법을 사용했다. 생각한 알고리즘은 우선 첫 블록을 놓을 위치는 모두 탐색한다. 이후, 탐색을 하면서 누적 점수 상위 6개 노드를 저장한다. 첫번째 탐색이 끝난 후, 저장된 노드들을 다시 함수를 호출하여 점수를 누적한다. 마찬가지로, VISBLE\_BLOCKS 까지 반복한다. 이후, 최종적으로 저장된 점수들을 비교하여, 가장 높은 점수를 가진 노드의 정보를 넘겨준다.

이때, 공간복잡도는 동일하며 시간복잡도의 경우 O 가 된다.

또한, 알고리즘이 점수를 조금 더 잘 낼 수 있도록 하기 위하여 여러 조건에 따른 가중치를 두었다. 여러가지 조건을 설정하고 변경하였지만, 최종적으로 결정된 고려한 상황은 다음과 같다. 아래 가중치는 recommend함수 modifed\_recommend함수 모두 적용된다.

1. 필드내의 전체 빈칸의 개수

2. 삭제되는 줄

3. 필드 높이의 합

4. 블록과 블록이 닿는 면적 , 바닥과 닿는 면적, 양옆 끝에 닿는 면적

5. 높이편차의 합

6. 현재 블록의 높이

가중치를 결정하기 위해 정말 많은 고민을 하였다. 처음에는 단순하게 블록을 놓았을 때 생기는 빈칸만을 고려하였다. 이후, 추가적으로 고려해야할 사항들을 생각해보고 추가하였다. 가중치의 경우 1~10까지 임의의 숫자를 랜덤으로 할당하면서 엄청난 시행을 반복하였다. 가중치를 파일에 저장하면서, 점수가 2만점이 넘기는 경우를 모두 저장하였다. 이후 각 앞자리를 정한 후, 뒤 4자리를 다시 정하기 위해 반복시행을 통해 값이 그나마 높은 경우를 할당하였다.

**2. modified\_recommend**

이 알고리즘의 경우 속도는 엄청나게 향상됨을 알 수 있다. 하지만, 얻어지는 점수의 경우 확연하게 차이가 난다.

**텍스트이(가) 표시된 사진

자동 생성된 설명**

위 그림은 일반 recommend함수를 통해 모든 경우를 탐색한 경우이다. VISIBLE\_BLOCKS는 3이다.

텍스트이(가) 표시된 사진

자동 생성된 설명

위 그림은 일반 modified\_recommend함수를 통해 모든 경우를 탐색한 경우이다. VISIBLE\_BLOCKS는 3이다.

시간을 많이 들여 함수를 여러 번 수정하였지만, 아예 안죽는 경우는 없었다. Recommend\_play함수에서 blockdown함수를 호출하기 위해 alarm(1)을 ualarm(10000,0)으로 바꾸고 진행하였다.

결론적으로, 알고 있는 정보량과 점수는 비례하며, 정보량과 시간은 반비례함을 알 수 있다. 두 함수는 VISIBLE\_BLOCKS이 1의 값을 가질 경우 같은 시간복잡도를 가지며, 얻어지는 결과 역시 비슷하다. 또한, 기존 AddtoBlock과 DeleteLine 함수만을 이용하여 추천시스템을 구현했을 때에 비해 위 가중치 시스템을 이용해 구현한 추천시스템이 훨씬 좋은 위치를 추천해주었다.

**3. 습득 내용 및 평가**

개인적으로 정말 재밌고 열심히 한 프로젝트이다. 자료도 열심히 찾아보고 어떻게 하면 테트리스 Ai가 죽지 않고 게임을 할 수 있을지 고민하며 나름대로 가중치를 찾기 위해 노력하였다. 유전적 알고리즘이나 , 머신러닝 등 여러 자료를 찾아보며 어떤식으로 컴퓨터가 최적의 값을 찾는지 어느정도 이해하고 알게되었다. 또한, BFS, DFS 등의 사용법 역시 잠시나마 사용해보며 어느정도 익숙해졌다.

**4. 참고문헌**

Tetris AI . (April 14, 2013). https://codemyroad.wordpress.com/2013/04/14/tetris-ai-the-near-perfect-player/.