# 10주차 결과보고서

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

**1. 실험시간에 작성한 프로그램의 알고리즘과 자료구조를 요약하여 기술하시오. 완성한 알고리즘의 시간 및 공간 복잡도를 보이시오.**

Modified_recommend()의 경우 현재 블록과 다음 2개의 블록을 고려하여 재귀적으로 트리를 구성한다. 이 점은 기존과 똑같다. 그러나 각 배치마다 추가적으로 구현한 evaluateField()함수를 호출하여 필드 상태를 단순히 삭제된 줄의 개수로서 평가하는 것에서부터 벗어나고자 했다. 구멍의 개수, 울퉁불퉁함, 총 높이까지 고려하고자 했다. 이는 작성자 본인이 10주차 실습까지 만든 테트리스 게임을 직접 해보면서 휴리스틱하게 봤을 때도 굉장히 합리적이지 못한 위치를 추천하는 것을 피하고자 했기 때문이다. 또한, 현재 루프에서 최고 점수보다 100점 이상 낮으면 해당 배치는 건너뛰었다. 물론 끝까지 갔을 때 더 큰 점수가 나올 수가 있긴 하지만, 어차피 다음 3개의 블록까지 고려하는데 굳이 그런 경우까지 고려할 필요가 없다고 생각했다. 추천 트리 노드는 다음과 같이 구성했다.

```
typedef struct _RecNode {
        int level;
        int accumulatedScore;
        char recField[WIDTH];
        struct _RecNode **child;
} RecNode;
```

기존의 char field[HEIGHT][WIDTH] 대신에 char recField[WIDTH];를 사용하였다. 각 열의 최고 높이만 저장해서 220byte에서 10byte까지 메모리를 줄일 수 있었다. 그리고 평가 함수에서는 높이를 기반으로 다시 2D로 복원해서 penalty를 계산했는데, 이 과정에서 hole 계산의 정확도는 굉장히 떨어지게 된다는 단점이 존재한다. 블록 종류마다 회전 수가 다른데 평균적으로 2.5라고 하고, WIDTH는 최대 10이기에 곱하면 25라고 볼 수 있다. 트리의 깊이는 3이기에 25^3은 15625이다. 그렇지만 점수 가지치기로 최대 반 정도 감소 한다고 봤을 때 7500 ~ 15000의 시간복잡도를 가진다고 볼 수 있고, 빅오로 표기하자면 O((R * W )^D)라고 볼 수 있다. 공간 복잡도의 경우 모든

노드를 다 생성한다면 15625개의 노드가 나올 것이고, RecNode는 24byte이기에 최악의 375KB가 쓰일 것이다.

**2. 모든 경우를 고려하는 tree 구조와 비교해서 어떤 점이 더 향상되고, 어떤 점이 그렇지 않은지 아울러 기술하시오**

모든 경우를 고려하던 경우와 비교하자면 나쁜 배치는 조기에 포기하여 50~70%로 줄일 수 있다. 그러나 local 루프에서 최고 점수와 비교할 때 쓰이는 파라미터 값 조절이 어려우며 최적해를 놓칠 수 있다. 또한 필드를 압축했다. 각 col별 높이만 저장하기에 10byte로 크기를 줄일 수 있다. 그러나 평가를 할 경우, 다시 2D로 복원해야 하고, 구멍 개수 평가의 정확도는 떨어지게 된다. 또한, 각 블럭 별로 중복인 경우의 수를 체크해 이를 제거했다. 이를 통해 불필요한 탐색을 줄일 수 있고, 앞서 서술했듯 단순히 삭제되는 줄의 개수 뿐만 아니라 울퉁불퉁함, 높이까지 고려하도록 만들어서 장기적으로 좋은 수에 놓도록 휴리스틱한 접근법을 사용하였다.

**3. 본 테트리스 프로젝트를 통해 습득한 내용을 한 내용이나 느낀 점을 기술하시오.**

트리 구조를 짤 때 메모리를 해제하는 것 및 포인터를 다시 복습하게 되었다. 알고리즘 최적화 기법 중 가지치기 및 조기 종료, 정보 최소화를 습득할 수 있었다. 또한, 계산 정확도와 시간, 메뢰 간의 여러 trade off가 있을 수 있다는 것을 느끼게 되었다. 막학기에 피하고 피해 컴실 1과 운영체제를 듣는 것은 굉장히 비합리적인 선택이었다고 생각이 들기도 했다...