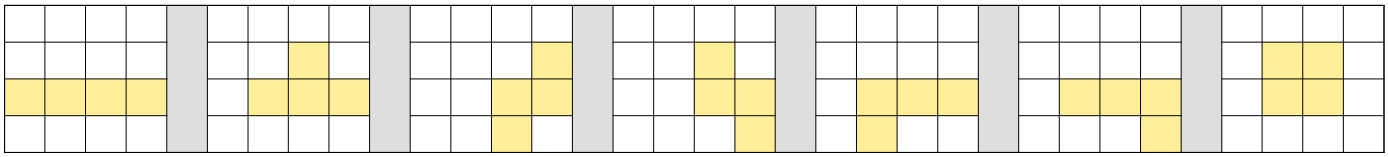
10주차 예비보고서

전공: 신문방송학과 학년: 3학년 학번: 20191150 이름: 전현길

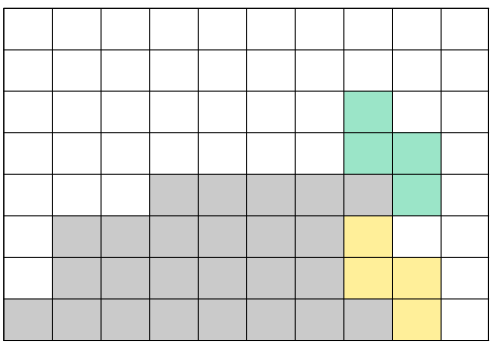
1. 테트리스 프로젝트 3주차에 구현하는 추천 기능에 대한 교재 내용을 읽어보고, 어떤 원리로 작동되는지 설명하시오. 그리고 추천 기능을 구현하는 tree의 장(효율성), 단점(비효율성)을 기술하시오.

교재의 내용에 따르면, 추천 시스템이란 '미래의 경우의 수를 고려하여 가장 높은 가중치를 가진 선택'을 고려하는 시스템이다. 테트리스의 경우 추천 시스템은 미래의 블록을 고려하여 가장 높은 점수를 얻을 수 있는 블록을 추천하는 시스템이라고 생각할 수 있다.

우리가 만들 테트리스의 추천 시스템은 '확실한 미래'를 고려하는데, 이는 연속적인 다음 블록에 대해 가능한 모든 경우의 수를 고려함으로써 가장 효율적인 선택을 하는 방식이다.



테트리스에 존재하는 블록은 위 7개이며, 각 블록마다 회전수가 최대 4가지 존재한다. 4번 블록의 경우 회전했을 때 모양이 총 2가지 존재하고, 하나는 가로세로 3*2칸이므로 가로 10칸의 필드를 고려하면 최소 8가지 경우의 수가 존재하며, 다른 하나는 가로세로 2*3칸이므로 가로 10칸의 필드를 고려했을 때 최소 9가지 놓일 수 있는 경우의 수가 존재한다. 단, 17가지 경우의 수는 최소 범위로 **필드의 형태에 따라서 충분히 경우의 수가 증가할 수 있다. 아래와 같은 경우에, 블록은 가로 2칸의 한 범위 안에 2칸 놓일 수 있다.**



1개의 블록에 대한 추천 시스템은 해당 경우의 수를 모두 고려한 뒤 가장 높은 점수를 받을 수 있는 블록의 위치를 화면에 보여준다. 2개, 3개의 블록에 대한 추천 시스템을 개발할 경우엔, 마찬가지로 2개, 3개 블록에 대한 경우의 수를 모두 고려한 뒤 가장 높은 점수를 받을 수 있는 블록의 위치를 보여준다.

추천 기능을 구현하는 가장 기초적인 tree 구조는 BLOCK_NUM개의 블록에 대해 가능한 경우의 수를 완전탐색함으로써 가장 높은 점수를 받을 수 있는 선택을 한다.

이러한 tree 구조의 **장점**은 최소한 n개의 블록에 대해 추천 시스템의 선택이 최선임이 보장된다는 것이다. 또한 예측할 수 없는 다양한 필드 상황에 대해 최선의 블록 위치를 고려하기 위해 복잡한 로직을 설계할 필요가 없다. 따라서 개발 측면에서 상당히 경제적이다.

반면에 **단점** 역시 여러 가지 있다. **첫 번째**는 BLOCK_NUM개의 블록에 대해서 최고의 점수를 제공하는 선택이 장기적으로 보았을 때에도 최선의 선택일지는 알 수 없다는 점이다.

예를 들어, 우리가 개발한 테트리스 게임은 한 번에 지워진 블록의 수를 제곱한 값에 100을 곱해 점수를 계산한다. 따라서 게임 플레이 시간이 정해져 있을 때 고득점을 얻기 위한 가장 좋은 전략은 가로 한 칸을 남기고 4줄을 채운 뒤 세로로 긴 블록이 나왔을 때 한꺼번에 없애는 것이다. 하지만 BLOCK_NUM 수가 너무 적다면 단기적인 미래를 주로 고려하기 때문에, 이후의 1600점을 포기하고 현재의 100점, 400점을 노리는 선택이 이뤄지기 쉽다.

두 번째는 시간 복잡도, 공간 복잡도 측면에서 굉장히 비효율적이라는 점이다. 교재에 설명되었던 대로 최악의 경우 한 블록당 34개의 경우의 수가 발생할 수 있으므로(앞서 보였듯 그보다 더 많은 경우의 수가 발생할 수도 있다), 시간 복잡도, 공간 복잡도는 O(34^BLOCK_NUM)이다.

따라서 고작 BLOCK_NUM을 5로 늘리더라도 계산해야 할 경우의 수는 45,435,424가지이다. 뿐만 아니라 경우의 수 하나당 10*22 크기의 필드를 저장하므로, 필드만 따져도 대략 880B의 메모리가 필요하고 이를 경우의 수에 곱하면 약 40GB의 메모리가 된다.

2. Tree 구조의 비효율성을 해결할 방법에 대해서 2가지 이상 생각하고, 그 아이디어를 기술하시오.

점수를 산정하는 가중치에 대한 조정, 경우의 수를 계산하는 로직에 대한 조정, 계산하는 양에 대한 조정, 경우의 수를 줄일 적절한 논리에 따른 pruning(가지치기), 메모리의 최적화 등을 통해 이러한 비효율성을 해결할 수 있다.

앞서 제시한 단점 중 첫 번째에 대한 해결책을 생각해보면, 크게 두 가지 해결책을 고려할 수 있다. 가장 먼저 떠오르는 해결책은 계산하는 BLOCK_NUM의 수를 증가시키는 것이다. 모델이 더 장기적인 미래를 예측하게 될수록, 당장의 미래만을 고려하는 등의 단기적인 선택을 피하게 될 것이다. 그러나 이는 trade-off가 있는 방법으로 O(34^n)의 시간, 공간 복잡도에 따른 연산 회수/메모리 증가 문제와 필연적으로 직면해야 한다.

다음으로는 가중치를 조정하는 방식을 생각해볼 수 있다. 먼저 줄을 1줄씩 지우거나 2줄씩 지우는 경우 계산되는 점수를 줄이는 방식을 생각해 볼 수 있을 것이다. 필드의 한 줄에 블록이 한 칸만 남아 있는 상태를 유지할 경우 점수 산정에 가산점을 주는 방식도 생각해볼 수 있다. 다만 이러한 가산점들이 3줄, 4줄을 한꺼번에 삭제할 때 얻을 수 있는 점수에 비해 눈에 띄게 높다면 블록을 지우지 않고 자살하는 플레이가 발생할 수 있으므로 면밀한 조정이 필요할 것으로 보인다.

혹은 경우의 수를 계산하는 로직 자체를 조정해볼 수도 있다. 예를 들어 필드에 블록이 3줄 이하로 존재한다면 추천 시스템이 필드의 가장 왼쪽 부분에 블록을 두는 경우의 수를 제거하도록 하고, 필드에 놓인 블록이 4줄 이상으로 증가하면 그 때 필드의 가장 왼쪽 칸에 블록을 두도록 만들 수 있다.

적절한 논리에 따른 pruning은 경우의 수를 줄이기 위한 방법이다. 교재에는 점수가 낮은 가지를 제거하는 방식이 소개되었는데, 이 경우 가장 좋은 점수를 얻을 수 있는 플레이 시퀀스를 고려하지 못하는 문제가 발생했다. 이러한 문제를 근본적으로 해결할 만한 특별한 방법을 생각해내진 못했지만, 경우의 수가 납득할 만큼의 숫자일 때까지는 완전 탐색하되, 경우의 수가 34^3가지 이상으로 증가할 때마다 가지치기를 수행하는 방식을 고려해 보았다.

메모리의 최적화는 교재에서 제공되었듯이 필드의 가로*세로 칸을 모두 고려하는 대신 높이만을 저장하는 방식 등으로 이뤄질 수 있다.