CSE3013 (컴퓨터공학 설계 및 실험 I)
PRJ-3 테트리스 프로젝트 3주차 예비 보고서

서강대학교 컴퓨터공학과 박수현 (20181634)

서강대학교 컴퓨터공학과

1 목적

추천 시스템의 동작 원리를 이해하고 단점을 보완할 방법을 제시한다.

2 추천 알고리즘

현재 상태를 루트로 갖는 트리를 만든다. 현재 블록과 다음 블록을 이용해 첫 번째 블록을 놓는 모든 위치와 회전 상태에 대해 자식 노드들을 만든다. 그 자식 노드들에서 두 번째 블록을 놓는 모든 위치와 회전 상태에 대해 자식 노드를 만든다. 그리고 트리를 탐색하면서 가장 높은 점수를 도출하는 위치에 다음 블럭을 놓는다.

간단히 말하면, 추천 알고리즘은 가능한 미래의 상태 공간을 결정 트리decision tree로 만들어 완전 탐색해 최적의 위치를 찾아 블록을 놓는다.

2.1 결정 트리의 장점

결정 트리를 사용해 추천 알고리즘을 작성할 경우 볼 수 있는 미래의 상태 공간을 전부 탐색할 수 있다. 따라서 현재 상황에서 최선의 결정이 가능하다.

2.2 결정 트리의 단점

회전해서 같아지는 상태를 고려하지 않고, 블록을 항상 필드 스택 위에 쌓는다(끼워 넣지 않는다)고 가정한다면, 블록 크기에 따라 다르지만 각각의 x 좌표와 회전 상태 4개에 따라 최대 34개의 상태가 가능하다. 따라서 추천 알고리즘이 다음 n개의 블록을 고려한다고 가정하면 상태마다 그 다음 상태는 34개가 가능하므로, 이 때 결정 트리의 총 노드 수는

$$\sum_{k=0}^{n} 34^k = \frac{34^{n+1}-1}{33}(\text{개})$$

이고, 이 때의 공간 복잡도는 $2^{\mathcal{O}(n)}$이고 시간 복잡도도 마찬가지로 지수 시간으로 비효율적이게 된다.

3 결정 트리의 단점의 개선 방법

Classic Tetris World Championship(CTWC)은 여러 플레이어가 NES[Nintendo Entertainment System] 테트리스에서 게임 오버 직전까지 달성한 점수로 겨루는 토너먼트 대회이다. NES 테트리스와 프로젝트에서 구현한 테트리스의 메커니즘은 대부분 같으며, NES 테트리스의 레벨 0 득점 기준에서 다음과 같은 차이가 있다.

조건	NES		본 프로젝트	
	점수	배율	점수	배율
1라인 클리어	40	×1.0	100	×1.0
2라인 클리어	100	×2.5	400	×4.0
3라인 클리어	300	×7.5	900	×9.0
4라인 클리어	1200	×30.0	1600	×16.0

NES 버전과 본 프로젝트 모두 많은 라인을 동시에 클리어할 경우 가산점을 부여하는 것을 확인할 수 있다. 특히 본 프로젝트에서는 1라인 클리어를 16번 하면 4라인 클리어를 1번 하는 것과 같은 점수를 얻게 되는데, 이런 경향이 더 심한 NES 테트리스의 경우 CTWC 결승 플레이어들은 왼쪽 9줄에는 블럭을 최대한 구멍 없이 쌓고, 남겨 둔 오른쪽 1줄에 I 블록을 끼워넣음으로서 4라인 클리어를 최대화시키는 전략을 사용하는 것을 확인할 수 있다.[1]

하지만 다음 조각의 개수가 현저히 적을 때 점수를 기준으로 블럭들의 위치를 판단하면 당장 줄을 지울 수 있는 1라인 클리어를 자주 수행할 것이다. 이 점에 착안하면 현재 상태에서의 최선의 결정은 전체 게임에서 최선의 결정이 아니며, 현재 상태에서 최선의 결정을 하는 것이 그렇게 중요하지 않다는 것을 알 수 있다.

3.1 점수 효율 개선 - 발견법[heuristic]적인 알고리즘 사용

블록을 놓았을 때 얻을 수 있는 점수를 고려하지 않고, 현재 필드의 상태를 계산하는 기준을 마련해 이를 기준으로 상태 공간의 노드들을 평가하도록 한다. 예를 들어, 클래식 테트리스 고수 플레이어들의 패턴을 분석하여 다음과 같은 기준 등을 선형 결합해 보드 점수를 매길 수 있겠다.

- 위가 덮인 구멍을 만드는 것은 비효율적이므로, 점수를 상당 부분 감산한다.
- 인접한 두 열과의 높이 차이의 절댓값이 클수록 구멍을 만들지 않고 놓을 수 있는 블록은 한정적이므로, 인접한 열들의 높이 차이의 절댓값의 합이 클수록 점수를 감산한다.
- 쌓아 둔 높이가 적당히 높아야 I 조각이 나왔을 때 4라인 클리어를 할 수 있으므로, 높이가 어느 정도 높을수록 점수를 가산한다.

- 쌓아 둔 높이가 너무 높으면 게임 오버의 가능성이 있으므로, 높이가 너무 높다면 점수를 상당 부분 감산한다.
- 보드 높이가 적당하다면, 4줄이 채워져 있는 경우 점수를 상당 부분 가산한다. 보드 높이가 너무 높다면, 게임 오버를 방지하기 위해 1줄이 채워져 있는 경우 점수를 상당 부분 가산한다.

3.2 시간 효율 개선 - 가지치기pruning

이렇게 다음 상태들에 대해 발견법heuristic적인 알고리즘을 적용해 계산한 점수에 대해, 현재 탐색하고 있는 트리 레벨에서 s개의 상태 공간만을 남기고 나머지 상태 공간을 전부 버린다. 그러면 각 레벨마다 34^k개가 아닌 최대 s개의 상태 공간만 체크하게 되며, 이 때 체크하는 총 노드 수는

$$\sum_{k=0}^{n} \min\left\{34^k, s\right\} \leq sn + 1(\text{개})$$

이고, 각 레벨마다 노드를 점수 기준으로 정렬해 상위 n개를 택하는 데 선형 로그 시간이 소요되므로 지수 시간 탐색 시간을 이차 로그 시간 $\mathcal{O}\left(s^2 n \log s\right)$으로 줄일 수 있게 된다.

3.3 공간 효율 개선 - 동적 트리 구성

너비 우선 탐색Breadth-First Search를 이용하면 트리의 노드들도 전부 만들 필요는 없다.

우선 루트에서 다음 블록을 놓은 상태로 가능한 모든 노드를 만들어 두고, 현재 레벨의 점수를 전부 계산하고 가지치기를 한 후 다음 레벨로 진행하지 않는 상태를 나타내는 노드들은 전부 메모리에서 지운다. 그리고 지워지지 않은 노드들에 대해서 다음 블록을 놓은 상태의 노드들을 만들고, 그 레벨에 대해서 가지치기를 하고, 가지치기당한 노드들을 메모리에서 지우고, 등을 레벨마다 반복한다.

각 레벨마다 $\min\left\{34^k, s\right\}$개의 노드만이 남게 되므로 공간 복잡도도 $\mathcal{O}(sn)$으로 줄일 수 있다.

3.4 공간 효율 개선 - 비트마스킹bitmasking

지금은 각 칸에 16비트 `short`의 공간을 할당하고 있으나, 각 칸의 색상 정보를 버리면 각 칸은 채워져 있는 상태와 채워져 있지 않은 상태만 필요하므로 1비트면 충분하다. 총 220비트가 필요하므로, `unsigned int` 7개를 사용하는 구조체를 만들어 필드를 row-major order로 저장할 수 있겠다.

참고문헌

[1] Classic Tetris, *16 Y/O UNDERDOG vs. 7-TIME CHAMP - Classic Tetris World Championship 2018 Final Round*. https://www.youtube.com/watch?v=L_UPHsGR6fM. Retrieved 19 May 2019.