전공 : 컴퓨터공학과 학년 : 4 학번 : 20212020 이름 : 박민준

1. 실습 시간에 작성한 프로그램의 알고리즘과 자료구조를 요약하여 기술하시오. 완성한 알고리즘(추가 구현하게 되는 효율성을 고려한 tree도 포함)의 시간 및 공간 복잡도를 보이시오.

1) 기본 추천 알고리즘 (recommend 함수)

텍스트, 스크린샷, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

텍스트, 스크린샷이(가) 표시된 사진

자동 생성된 설명

- 이중 for문을 사용하여 트리 구조를 생성한다. 현재 블록과 다음 두 개의 블록을 고려하여 모든 가능한 위치와 회전을 평가한다.

- 각 위치에서 블록을 놓았을 때의 누적 점수를 계산하고, 최적의 경로를 탐색하여 가장 높은 점수를 얻을 수 있는 경로를 찾는다.

2) 개선된 추천 알고리즘 (modified\_recommend 함수 with updateFieldHeight 함수)

텍스트, 스크린샷, 폰트, 디스플레이이(가) 표시된 사진

자동 생성된 설명

- 위 updateFieldHeight 함수는 현재 필드에서 각 열(column)마다 블록이 쌓인 높이를 계산하여 저장한다. 이를 통해 필드의 전체 상태를 압축된 형태로 표현할 수 있다. 이 정보는 추천 시스템의 메모리 사용량을 줄이는 데 활용된다.

- height[WIDTH]: 각 열마다 블록이 쌓인 높이를 저장하는 1차원 배열이다.

- recField[HEIGHT][WIDTH]: 필드 상태를 저장하는 2차원 배열이다. 블록이 있는 위치는 1, 비어 있는 위치는 0으로 표시된다.

- 해당 함수의 알고리즘은 다음과 같다. 먼저, 각 열의 높이 정보를 저장할 배열 height를 초기화한다. 이 때, 배열의 크기는 필드의 너비 WIDTH와 동일하다.

- 그 후, 각 열마다 가장 위에 있는 블록의 위치를 찾기 위해 각 열의 맨 아래부터 시작하여 위로 올라가며 블록을 찾는다. 블록을 찾으면 해당 위치에서 필드의 맨 아래까지의 거리를 해당 열의 높이로 설정한다.

- 마지막으로 계산된 높이 정보를 height 배열에 저장한다.

텍스트, 스크린샷, 소프트웨어, 디스플레이이(가) 표시된 사진

자동 생성된 설명

텍스트, 스크린샷, 소프트웨어, 멀티미디어 소프트웨어이(가) 표시된 사진

자동 생성된 설명

- modified\_recommend 함수는 데이터 단순화 (Data Simplification) 방법을 이용해 tree 구조의 비효율성을 해결한 함수이다. 해당 함수는 필드 상태를 저장할 때 전체 필드가 아닌 필드의 높이만을 저장하여 공간 효율성을 높인다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

- recommend 함수에 사용된 RecNode 구조체의 구조는 위와 같다.

- level: 현재 노드의 트리 레벨.

- accumulatedScore: 현재까지의 누적 점수.

- recField: 블록이 놓여진 후의 필드 상태.

- child: 자식 노드들을 가리키는 포인터 배열.

- curBlockID: 현재 블록의 ID.

3) 시간 및 공간 복잡도

- recommend 함수에서 각 블록은 4가지 회전 상태를 가질 수 있으며, 가로로 이동할 수 있는 위치는 WIDTH + 2 (블록의 크기 고려)이다.

- 블록이 놓일 수 있는 모든 위치를 평가하기 때문에, 각 레벨에서 최대 4 \* (WIDTH + 2) 개의 노드를 생성한다.

- 트리의 깊이가 VISIBLE\_BLOCKS라면, 시간 복잡도는 O((4 \* (WIDTH + 2))^VISIBLE\_BLOCKS)이다.

- 또한, 각 노드는 필드 상태와 몇 가지 추가 정보를 저장한다.

- 최대 노드 수는 (4 \* (WIDTH + 2))^VISIBLE\_BLOCKS이므로, 공간 복잡도는 O((4 \* (WIDTH + 2))^VISIBLE\_BLOCKS \* 필드 크기)이다.

- modified\_recommend 함수는 필드 높이 정보만 저장하여 공간 사용을 줄인다.

- 각 노드가 필드 높이 정보를 저장하므로, 각 노드의 공간은 O(WIDTH)이다.

- 따라서, 공간 복잡도는 O(k \* (4 \* (WIDTH + 2))^VISIBLE\_BLOCKS \* WIDTH)이다. 여기서 k는 가지치기 후 남은 경로의 비율을 의미한다. 만약 가지치기 없이 모든 경로를 탐색할 경우, 공간 복잡도는 O((4 \* (WIDTH + 2))^VISIBLE\_BLOCKS \* WIDTH)이다.

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

1) 향상된 점

- 모든 경우를 고려하는 트리 구조는 기하급수적으로 많은 노드를 생성하여 많은 시간을 소모한다. 개선된 알고리즘은 가지치기(pruning) 기법을 통해 고려할 경우의 수를 줄임으로써 시간 효율성을 높일 수 있다. 가지치기를 통해 더 깊은 레벨로의 재귀 호출 횟수가 줄어들어, 재귀적으로 호출되는 함수의 수가 줄어들고 이에 따라 시간 복잡도도 감소한다.

- 또한, 데이터 단순화를 통해 필드의 높이 정보만을 저장하여 메모리 사용량을 줄일 수 있다. 이는 불필요한 세부 정보를 저장하지 않음으로써 메모리를 더 효율적으로 사용할 수 있고, 전체 필드 상태를 저장하는 것보다 훨씬 적은 공간을 차지하게 된다.

2) 향상되지 않은 점

- 가지치기 기법을 사용할 경우, 일부 경로를 미리 제거함으로써 최적의 경로를 놓칠 가능성이 있다. 이는 전체 경로를 평가하는 것이 아니기 때문에, 최종적으로 추천된 경로가 반드시 최적이 아닐 수 있다. 만약 가지치기 기준이 부정확할 경우, 실제로 높은 점수를 얻을 수 있는 경로를 제거하는 오류가 발생할 수 있다.

- 또한, 가지치기 및 데이터 단순화 기법을 구현하는 과정에서 알고리즘이 복잡해지고, 이를 유지보수하는 데 추가적인 노력이 필요하다. 가지치기와 데이터 단순화로 인해 발생하는 예상치 못한 오류나 버그를 디버깅하는 데 어려움이 있을 수 있다.

3. 테트리스 프로젝트 3주 과정을 통해 습득한 내용이나 느낀 점을 기술하시오.

- 테트리스 프로젝트를 통해 트리 구조와 재귀적 알고리즘에 대한 깊은 이해를 하게 되었다. 특히, 트리를 사용하여 게임 상태를 관리하고, 재귀적 방법으로 최적의 블록 배치를 찾는 과정이 가장 흥미로웠다. 이러한 기법들은 게임뿐만 아니라 다양한 문제 해결에 적용될 수 있는 강력한 도구임을 깨닫게 되었다. 또한, 프로젝트에서 구현한 데이터 단순화(Data Simplification) 기법은 공간 효율성을 크게 향상시키는 방법을 학습하는 좋은 기회였다.

- 이론적으로 배운 알고리즘을 실제 문제에 적용하는 과정에서 예상치 못한 어려움이 많았다. 특히, 테트리스처럼 실시간으로 빠르게 진행되는 게임에서는 단순히 알고리즘의 정확성뿐만 아니라, 실행 속도와 메모리 사용량도 고려해야 했다. 이러한 제약 조건을 맞추기 위해 알고리즘을 최적화하는 방법을 배울 수 있었다.

- 어렸을 때, 게임으로만 접했던 테트리스 게임을 직접 구현해보니 신기하기도 했고, 약 800줄이 넘는 코드이다 보니 프로젝트가 복잡해짐에 따라 코드의 가독성과 유지보수성이 매우 중요하다는 것을 깨닫게 되었다. 비록 이번 테트리스 프로젝트는 혼자 구현하였지만, 나중에 팀 프로젝트를 진행하게 된다면 내가 작성한 코드를 다른 팀원들이 이해하고 수정하기 위해서는 코드가 명확해야 하고 주석이 잘 달려있어야 한다는 생각이 들었다. 이를 통해 클린 코드 작성의 중요성을 다시 한번 느끼게 되었다.