CSE3013 컴퓨터공학설계및실험I: 테트리스 프로젝트

담당교수: 서강대학교 컴퓨터공학과 김승욱

1. 설계 문제 및 목표

ncurses 라이브러리가 제공하는 리눅스 터미널 상에서의 GUI를 이용하여 테트리스 게임 프로그램을 제작하고 여기에 랭킹 시스템과 블록 배치 추천 기능을 추가 구현한다.

테트리스 게임은, 블록을 90도씩 반시계 방향으로 회전하거나 세 방향(좌, 우, 하)으로 움직여 필드에 쌓으면, 빈틈없이 채워진 줄은 지워지고 그에 따른 스코어를 얻어 결과적으로 가장 높은 최종 스코어를 얻는 것이 목적이다.

프로그램을 실행하면 사용자는 메뉴를 선택할 수 있다. 1번을 선택하면 테트리스 게임을 플레이할 수 있고, 2번을 선택하면 랭킹 정보를 확인할 수 있고, 3번을 선택하면 블록 배치 추천 기능을 따라 플레이되는 모드로 테트리스 게임이 실행되고, 마지막으로 4번을 선택하면 프로그램이 종료된다.

1주차에서는 테트리스 게임의 기본적인 1번 Play 기능을 구현하는 것을 목표로 한다. Play 기능은 사용자가 직접 테트리스 게임을 실행하면서 블록의 줄을 모두 채워서 지우거나 다른 블록과 맞닿을 때 점수를 얻는다. 블록은 화면에 현재 블록, 다음 블록, 그리고 그 다음 블록으로 총 세 개가 보여야 한다. 매초마다 현재 블록이 하나씩 아래로 내려오며, 사용자가 키보드로 원하는 명령을 입력하면 그에 맞는 명령을 시행한다. 단, 명령이 실행되는 상황에서 받아들여질 수 있는지를 고려해야 한다. 또한 현재 블록에서 테트리스 필드 아래에 놓을 때 어느 위치에 놓이는지를 알려주는 그림자를 표시한다.

2주차에서는 처음 프로그램을 실행했을 때 미리 랭킹 정보를 읽어와 있어야 한다. 1번 기능에서 랭킹과 점수를 파일에 기록하고 원하는 정보를 입력하면 그에 맞는 랭킹 정보를 출력하는 2번 기능을 구현하는 것을 목표로 한다. 사용자가 1번 Play 기능을 수행한 후 Game Over가 되었을 때까지 얻은 점수를 rank.txt에 파일에 기록한다. 2번 기능에서 1번을 선택하면 원하는 순위의 범위에 해당하는 랭킹 정보를 출력한다. 2번을 선택하면 찾고자 하는 이름을 입력했을 때 그 이름을 갖고 있는 사용자의 랭킹 정보를 출력한다. 3번을 선택하면 지우고자 하는 순위를 랭킹 정보와 rank.txt에서 삭제해야 한다.

3주차에서는 1번 기능에서의 추천 시스템과 이에 따라 자동으로 테트리스 게임을 시행하는 3번 Recommend 기능을 구현하는 것을 목표로 한다. 1번 기능을 실행할 때 현재 블록과 다음 블록들의 정보를 참고하여 가장 높은 점수를 얻을 수 있는 경우를 찾아서 현재 블록을 어느 위치에 놓아야 할지 추천해야 한다. 3번 기능을 실행하면 1번 기능에서의 추천 시스템에 따라 사용자의 간섭 없이 프로그램이 자동으로 추천 위치에 블록을 놓아야 한다. 점수 대 실행 시간의 비율을 높이기 위해 3번 기능을 최적화해야 한다.

1.1 현실적 제한조건

이번 프로그램이 갖고 있는 한계는 최적화한 추천 시스템이 이상적이지 못하다는 것이다. 점수 대비 시간 효율을 향상시키기 위해 트리 가지치기를 사용했지만 이론적으로 가장 높은 점수를 얻는 경우라고 보장하기 어렵다. 일정 깊이 전까지 가장 누적 점수가 높은 5개의 노드만 남기고 나머지를 다 삭제한 후 남은 노드만 보고자 하는 깊이까지 탐색하여 가장 누적 점수가 높은 경우를 고른다. 그런데 만약 삭제하는 노드 중에서 다음 깊이까지 봤을 때 누적 점수가

가장 높은 경우가 존재한다면 이는 오히려 최적화한 것이 손해이다. 그래서 가중치를 사용해서 높은 점수를 얻을 수 있거나 Game Over를 최대한 피할 수 있는 경우에 높은 가중치를 부여해서 누적 점수의 편차를 늘렸다. 그래서 삭제하는 노드들 중에서 깊이 있게 더 봤을 때 누적 점수가 높은 나머지 5개 노드들의 점수를 초과하기 어렵게 만들었다.

이론과 실제 구현 사이에 차이가 있는 부분은 이진 탐색 트리(BST) 자료구조를 사용하여 랭킹 정보를 다루는 것이다. 연결 리스트를 사용하는 것보다 원하는 데이터를 탐색할 때 시간 복잡도를 줄이기 위해 이진 탐색 트리를 사용했다. 실제로 랭킹을 삽입 또는 삭제 자체의 시간 복잡도는 연결 리스트를 사용했을 때의 시간 복잡도인 $O(n)$에서 이진 탐색 트리의 최대 깊이에 해당하는 $O(\log n)$으로 줄였으나, 흐트러진 이진 탐색 트리의 랭킹 순위를 다시 정렬하는 과정이 불가피했다. 그래서 inorder 순으로 트리를 탐색할 때의 시간 복잡도가 $O(n)$에 근접하게 된다. 그러므로 이론상으로는 트리의 데이터를 다루는 데 걸리는 시간 복잡도를 줄이기 위해 이진 탐색 트리를 도입했으나, 정작 일부 기능을 수행할 때만 가능할 뿐 나머지 기능을 수행에서 시간 복잡도 상의 이득을 보지 못했다.

2. 요구사항

2.1 설계 목표 설정

<table>
<tr><th colspan="3">이번 프로젝트의 개괄적인 목표</th></tr>
<tr><td colspan="3">ncurses 라이브러리를 사용해서 linux 상에서 테트리스 게임의
기본적인 기능을 구현하고 랭킹 시스템과 추천 기능을 구현한다.</td></tr>
<tr><td rowspan="2">Play
기능</td><td colspan="2">블록을 매초마다 한 칸씩 아래로 이동시킨다.</td></tr>
<tr><td>요구사항 ①</td><td>BlockDown 함수를 1초마다 호출한다.</td></tr>
<tr><td colspan="3">블록의 줄을 모두 채워서 지우거나 다른 블록들과 맞닿을 때 점수를 얻는다.</td></tr>
<tr><td>요구사항 ②</td><td colspan="2">DeleteLine와 AddBlockToField 함수를 구현한다.</td></tr>
<tr><td colspan="3">다음 블록을 두 개 보여주고, 현재 블록을 내려서 점수를 얻은 후 다음 블록들을 현재 블록으로 앞당겨 오면서 맨 마지막 다음 블록을 임의로 생성한다.</td></tr>
<tr><td>요구사항 ③</td><td colspan="2">BlockDown 함수에서 다음 블록 배열을 앞으로 한 index씩 가져온다. 마지막 index의 블록 생성은 rand 함수를 사용한다. 블록이 7개이므로 0부터 6까지 중 임의의 수 하나를 생성해야 한다.</td></tr>
<tr><td colspan="3">사용자가 키보드로 명령을 입력하면 그에 맞는 작업을 수행한다. 단, 명령을 수행할 수 있는지 그 여부를 확인해야 한다.</td></tr>
<tr><td>요구사항 ④</td><td colspan="2">CheckToMove 함수를 호출하여 현재 블록을 키보드 방향의 명령에 맞게 변경 가능한지 확인하고, 만족하면 DrawChange 함수를 실행한다.</td></tr>
<tr><td colspan="3">현재 블록을 최대한 아래로 내렸을 때 어느 위치에 놓일 수 있는지 그림자로 표시한다.</td></tr>
<tr><td>요구사항 ⑤</td><td colspan="2">DrawShadow 함수에서 CheckToMove 함수를 사용하여 현재 블록이 최대로 내려왔을 때의 좌표를 알아내고 DrawBlock 함수를 호출한다.</td></tr>
</table>

rank 기능	Game Over 시 점수를 랭킹 정보에 입력한다.	
	요구사항 ⑥	newRank와 writeRankFile 함수를 구현해야 하며, Play 함수에서 Game Over가 되었을 때 두 함수를 실행한다.
테트리스 프로그램이 실행되었을 때 이미 랭킹 정보를 읽어 온 상태여야 한다.		
요구사항 ⑦	createRankList 함수를 구현하여 main 함수에서 이를 실행한다.	
원하는 순위 범위(또는 이름)에 해당하는 랭킹 정보를 출력한다.		
요구사항 ⑧	순위 데이터(또는 이름)를 가지고 inorder순으로 BST를 탐색해야 한다.	
원하는 순위에 해당하는 랭킹 정보를 삭제한다.		
요구사항 ⑨	순위 데이터를 가지고 탐색을 하면서 해당 순위 노드를 삭제하고, 랭킹 정보가 흐트러지지 않도록 다시 정렬한다.	
recommend play 기능	다음 블록 정보들을 참고하여 가장 높은 점수를 얻을 수 있는 경우를 수행하기 위해 현재 블록이 놓여야 할 위치를 추천한다.	
	요구사항 ⑩	트리를 사용하여 일정 깊이만큼 블록이 놓일 수 있는 경우의 수를 고려하는 recommend 함수를 구현한다.
3번 기능에서 추천 시스템을 바탕으로 추천 위치에 자동으로 현재 블록을 놓는다.		
요구사항 ⑪	Recommend 기능을 실행하면 추천 시스템에 의해 추천되는 위치를 BlockDown 함수에서 바로 사용한다.	
구현한 recommend 함수의 시간 및 공간 복잡도를 개선하여 최적화한다.		
요구사항 ⑫	트리의 크기를 줄이거나 효율적으로 탐색하는 방법을 고안한다.	

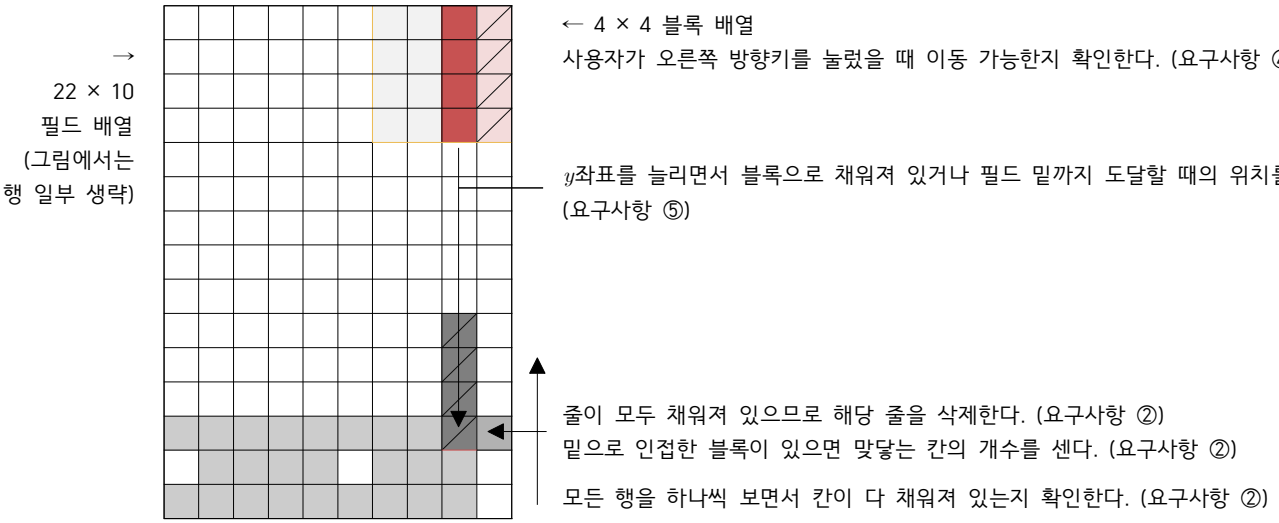
[표 1] 각 기능별 목표와 요구사항

2.2 합성

요구사항 ①을 구현하기 위해서는 signal.h에 존재하는 sigaction 시스템이 사용된다. 이는 특정 시그널 수신에 대해서 취할 액션을 설정하거나 변경하기 위한 것이다. 시그널이 발생했을 때 실행되는 함수를 sa_handler에 할당해야 한다. 그래서 play 함수가 실행될 때 handler 함수로써 BlockDown 함수로 설정해준다.

요구사항 ②, ④, ⑤에서는 현재 블록들이 놓인 필드의 데이터를 갖고 있는 배열과 4×4 블록 배열을 사용한다. 요구사항 ②에서 DeleteLine 함수를 구현하기 위한 알고리즘의 핵심 아이디어는 필드의 모든 행을 차례대로 탐색하면서 모든 열들이 채워져 있는지 확인하는 것이다. 또한 AddBlockToLine 함수와 요구사항 ④를 구현하기 위해서는 현재 블록의 위치와 그 주변에 인접한 필드 배열의 데이터를 확인해야 한다. 만일 현재 블록의 바로 밑에 블록이 존재하면 맞닿는 칸의 개수를 세서 몇 개를 닿는지 확인하여 점수에 반영해야 한다. 현재 블록의 위치에서 오른쪽 또는 왼쪽 칸에 이미 블록이 존재하거나 현재 위치의 오른쪽과 왼쪽이 필드의 양 끝 위치에 해당하면 사용자가 오른쪽 또는 왼쪽 방향키를 눌렀을 때 움직이지 못하도록 해야 한다. 또한 만일 위쪽 방향키를 눌러 블록을 회전하고자 하는데 회전했을 때의 블록이 필드 밖을 초과하거나 이미 그 자리에 블록이 존재하면 마찬가지로 명령을 거부해야

한다. 요구사항 ⑤에서는 현재 블록이 위치한 좌표에서 y좌표를 늘리면서 해당 칸이 비어 있는지 확인한다. 만약 칸이 다른 블록으로 채워져 있거나 필드의 밑까지 도달했으면 해당 칸 바로 위에 그림자를 그린다.



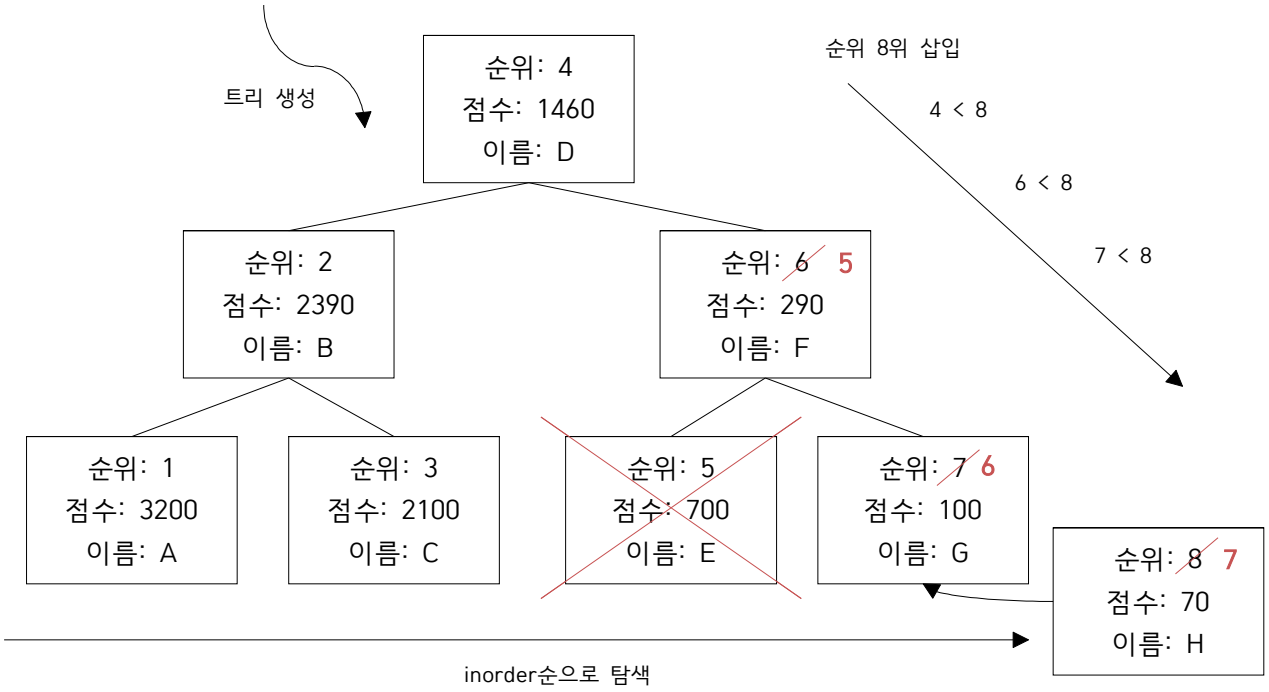
[그림 1] 요구사항 ② ~ ⑤의 수행을 만족하는 알고리즘

요구사항 ③에서는 현재 블록과 다음 블록을 저장하는 배열을 생성하여 사용한다. BlockDown 함수에 의해 현재 블록이 맨 밑에 그려지고 점수를 얻은 이후 for loop을 사용하여 블록 배열의 value를 한 index씩 앞으로 당겨온다. 마지막 index에는 rand 함수에 의해 생성된 임의의 수를 7로 나눴을 때의 나머지를 저장한다.

요구사항 ⑥에서 ⑨까지를 구현하기 위해 이진 탐색 트리(BST)를 사용했다. 이진 탐색 트리란 각 노드에 어떠한 값이 존재하고 그 값들 사이에 전순서가 있어야 한다. 이 프로그램에서는 BST에 점수가 가장 큰 1위부터 차례대로 저장해야 하므로 점수 값을 기준으로 노드의 왼쪽 서브트리에는 그 노드의 값보다 큰 값들을 지닌 노드들로, 오른쪽 서브트리에는 작은 값들을 지닌 노드들로 구성해야 한다. 각 노드들은 점수 값과 랭킹 순위, 이름을 저장해야 하며, 트리의 깊이를 최소한으로 줄이기 위해 한쪽으로 치우쳐 있지 않고 균형 있는 트리를 만들어야 한다. 그래서 우선 rank.txt의 데이터를 배열에 원소로 차례대로 저장한 후 데이터의 중간 index에 위치한 원소를 기준 노드로, 중간보다 왼쪽에 있는 원소들은 기준 노드의 왼쪽 자식으로, 중간보다 오른쪽에 있는 원소들은 기준 노드의 오른쪽 자식으로 만들어서 재귀적으로 이를 반복한다.

원하는 랭킹 정보 검색을 수행하기 위해서 트리 탐색 기능을 구현해야 한다. 검색하고자 하는 순위를 root 노드와 먼저 비교하고, 일치할 경우 이를 반환한다. 그렇지 않고 검색하고자 하는 순위가 root 노드보다 작을 경우 왼쪽 서브 트리에서 재귀적으로, root 노드보다 크거나 같은 경우 오른쪽 서브 트리에서 재귀적으로 검색을 수행한다. 마찬가지로 새로운 랭킹 정보의 삽입과 삭제도 검색을 통해서 수행하는데, 이 때 유의할 점은 전체 트리에서 랭킹 순으로 노드들이 정렬되어 있지 않을 수 있기 때문에 이를 다시 정렬해주는 작업을 필요로 한다. 이를 위해 모든 노드를 inorder순으로 탐색하면서 순위를 업데이트 한다.

1	2	3	4	5	6	7
3200	2390	2100	1460	700	290	100
A	B	C	D	E	F	G

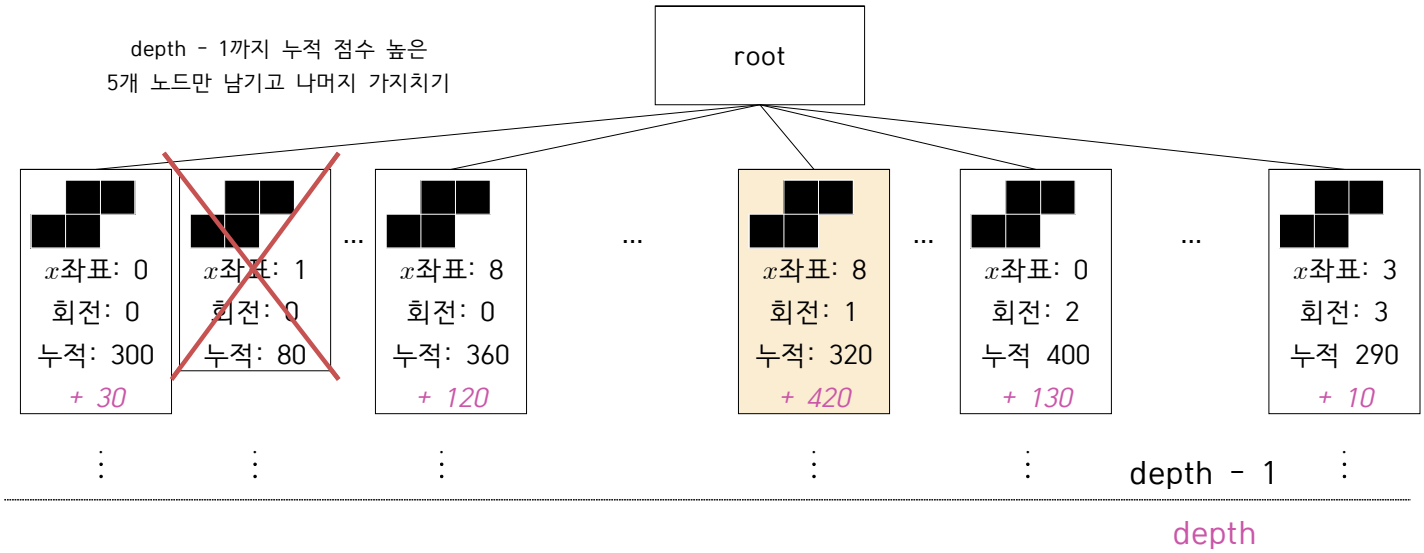


[그림 2] 요구사항 ⑥ ~ ⑨를 만족하는 이진 탐색 트리 구현 알고리즘

이진 탐색 트리를 사용한 각 요구사항 수행 예시 (※ [그림 2] 참조)
◎ 1위부터 3위까지 출력: inorder순으로 노드 탐색하면서 노드에 저장된 순위가 1이상 3이하인 것들 출력하기
◎ 8위 랭킹 정보 새로 삽입: 순위를 기준으로 이진 트리 검색하면서 삽입 가능한 위치 탐색하기
◎ 5위 랭킹 정보 삭제: 해당 노드 삭제하고 랭킹 다시 업데이트하기

요구사항 ⑩에서 ⑫까지는 자식 노드의 개수에 제한이 없는 트리 자료구조를 사용했다. 현재 또는 다음 블록들이 필드 내에 놓일 수 있는 경우의 수를 모두 고려하여 각각의 경우마다 놓였을 때의 누적 점수를 구한다. 일정 깊이의 수만큼 다음 블록들을 참고하여 각각의 위치에 놓이는 경우의 점수를 x좌표와 회전 형태에 대한 for loop을 사용하여 알아내고, recommend 함수를 재귀적으로 호출해서 다음 블록에 관하여 같은 과정을 시행한 후 일정 깊이까지 도달했을 때의 누적 점수를 반환한다. 각각의 경우들의 누적 점수 중에서 가장 높은 누적 점수를 찾고, 그 경우에서 현재 블록이 놓여야 하는 위치를 파악하여 해당 위치에 있을 때의 블록을 표시하거나 BlockDown 함수를 실행할 때 블록이 놓이는 위치를 해당 위치로 반영한다. 요구사항 ⑫에서는 위의 알고리즘과 자료구조를 사용했을 때 점수 대 시간의 비율을 최대한 끌어올려야 하는데, 이를 위해 트리 가지치기와 블록이 놓일 수 있는 중요한 경우에 큰 가중치를 부여하는 방법을 사용했다. 이 프로그램에서 사용한 트리 가지치기는 일정 깊이 한 단계 전까지 봤을 때 누적 점수가 높은 5개의 노드만을 남기고 나머지는 삭제한 후, 5개 노드들만 다시 일정 깊이 끝까지 도달해서 누적 점수를 구하여 가장 높은 점수를 얻을 수 있는 경우를 찾는 것이다. 가중치는 누적 점수를 구하는 과정에서 블록을 해당 위치에 놓았을 때 줄을 삭제하는 경우, 다른 블록과 맞닿는 경우, 필드 내 가장자리에 맞닿을 경우 순으로 높게 부여해

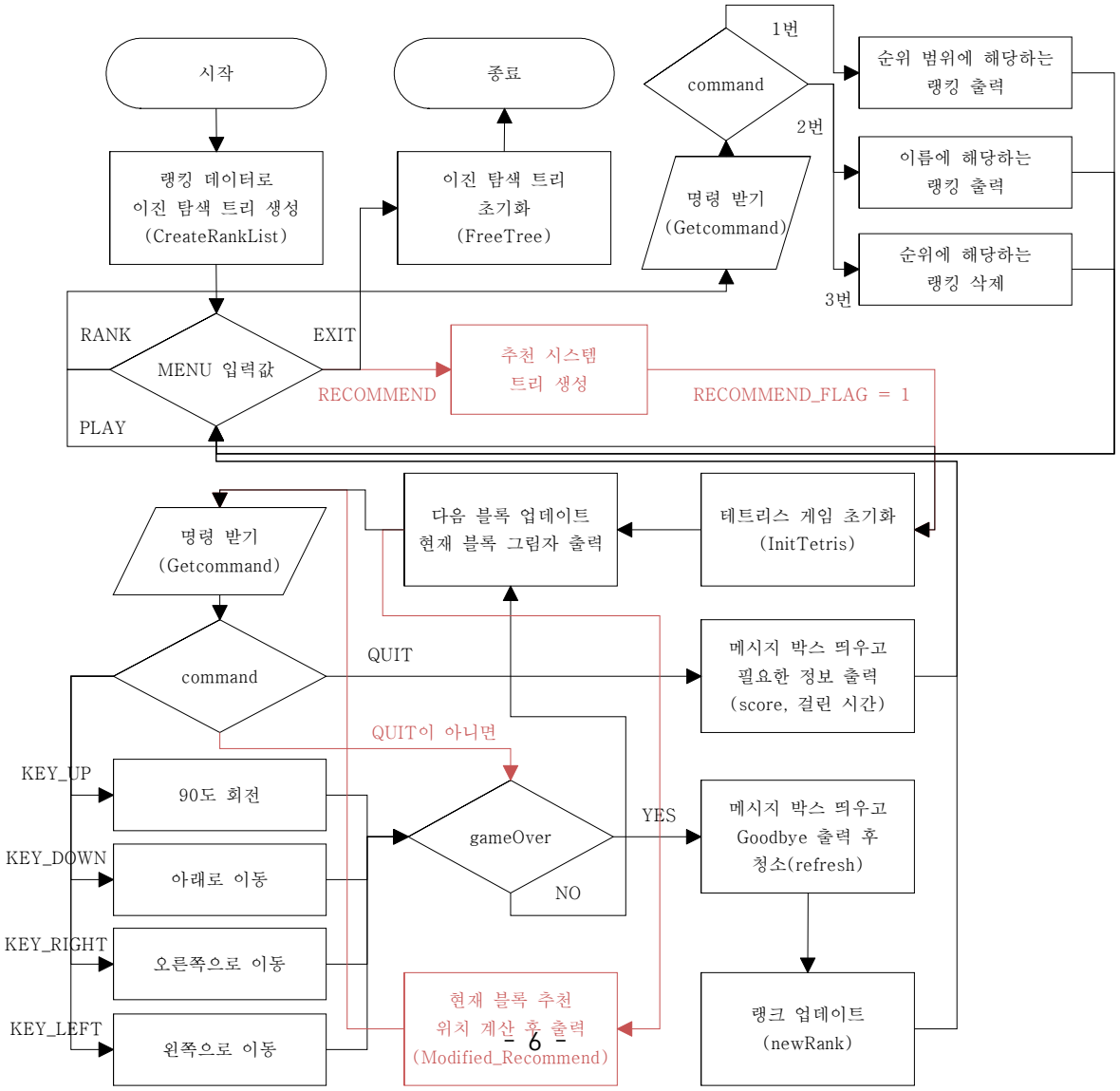
서 Game Over가 발생하는 상황을 피한다.



[그림 3] 요구사항 ⑩ ~ ⑫를 만족하는 트리 가지치기 구현 알고리즘

2.3 분석

[그림 4]는 이번 테트리스 프로그램의 전체적인 순서도를 나타낸다.



[그림 4] 테트리스 프로그램 Flow Chart

요구사항 ①을 수행하기 위해서는 BlockDown 함수를 signal의 handler 함수로 설정해준다. BlockDown 함수를 실행할 때마다 블록의 y좌표인 blockY가 하나씩 늘어나야 하므로 CheckToMove 함수를 호출하여 블록이 현재 위치에서 아래로 한 칸 더 움직일 수 있는지 확인해야 한다. 움직일 수 있으면 blockY를 하나 증가시켜주고 DrawChange 함수를 실행하며, 움직일 수 없으면 다음 블록의 정보를 한 index씩 앞으로 갱신하며 맨 다음 블록의 정보를 임의로 생성해야 한다. 요구사항 ① 자체는 블록이 아래로 이동가능한지 확인하는 조건문에서의 시간 복잡도이므로 $O(1)$이다. 공간 복잡도는 지역 변수를 생성하지 않고 오직 다른 요구사항에서 필요로 하는 함수들을 호출하면서 heap이 차지하는 비용이기 때문에 마찬가지로 $O(1)$이다.

요구사항 ②를 수행하기 위해서는 AddBlockToField와 DeleteLine 함수를 구현해야 한다. AddBlockToField 함수의 구체적인 알고리즘은 다음과 같다.

1) 블록 모양에 대한 정보를 가지고 있는 4×4 배열 block에 관하여 각 원소마다 값이 1인지 아닌지를 block 배열의 i좌표와 j좌표에 관한 for loop을 실행하여 확인한다.
2) 블록의 y좌표와 x좌표 값이 필드 내에 위치하는지 확인하고, 이를 만족하면 필드에 해당 칸이 차지된 것으로 업데이트한다.
3) 만약 블록의 바로 아래 좌표에 해당하는 칸이 필드의 밑바닥에 해당하거나 다른 블록이 이미 차지하고 있으면 닿은 것으로 인식하여 touched 변수를 증가시킨다.
4) for loop를 실행하여 모든 각 원소마다 2)~3)의 과정을 반복하고 touched 변수에 10을 곱하여 점수를 반환한다.

시간 복잡도는 block 배열의 i좌표와 j좌표의 이중 for loop에서 차지하는 비용에 의존하는데, 각 좌표는 최대 4를 넘기지 못하므로 시간 복잡도는 상수가 된다. 따라서 $O(1)$이다. 공간 복잡도는 필드 배열 자체가 전역변수이므로 함수를 실행하기 전부터 이미 차지하고 있는 공간 복잡도라고 가정할 때, 위의 함수 자체에서는 상수 개의 지역 변수 선언에서 드는 비용에 해당하므로 $O(1)$이다.

DeleteLine 함수의 구체적인 알고리즘은 다음과 같다.

1) 모든 줄을 살펴보기 위해 y좌표에 관한 for loop와 x좌표에 관한 for loop을 실행한다. x좌표 for loop를 실행할 때 해당 필드의 칸이 1로 설정되어 있지 않으면 x좌표 for loop를 벗어나고 다음 줄로 이동한다.
2) 만약 x좌표가 필드 오른쪽 끝까지 도달했으면 현재 위치한 y좌표에 해당하는 줄 위에 위치한 줄들을 한 칸씩 아래로 내린다. 이도 마찬가지로 for loop을 실행하여 필드 배열의 y좌표가 하나 작은 원소를 해당 y좌표에 해당하는 원소의 값으로 업데이트한다.
3) 첫 번째 줄은 무조건 삭제되므로 첫 번째 줄에 해당하는 필드 배열 원소들을 0으로 초기화한다.
4) for loop를 실행하여 모든 각 원소마다 1)~3)의 과정을 반복하고 touched 변수에 10을 곱하여 점수를 반환한다.

시간 복잡도는 필드 배열의 x좌표와 y좌표의 이중 for loop에서 차지하는 비용에 의존하는데, 각 좌표는 $WIDTH$와 $HEIGHT$을 넘지 못하므로 두 변수가 전역변수임을 가정할 때 upper bound는 $O(WIDTH \times HEIGHT)$이 된다. 공간 복잡도는 상수 개의 지역 변수 선언에

서 드는 비용이므로 $O(1)$이다.

요구사항 ③을 수행하기 위해서는 BlockDown 함수를 실행할 때 블록이 더 이상 아래로 내려올 수는 없지만 Game Over 상태가 아니면 다음 블록의 정보를 갖고 있는 nextBlock 배열의 원소들의 값을 하나씩 앞으로 복사하고, 맨 마지막 블록 원소는 난수를 생성하여 이를 7로 나눈 나머지를 저장해야 한다. 이후 새로 바뀐 다음 블록 정보를 화면에 반영하기 위해 DrawNextBlock 함수를 실행한다. DrawNextBlock에서는 4×4 배열인 block의 i좌표와 j좌표에 관한 for loop을 실행하면서 배열의 해당 칸이 1인 칸을 출력한다. 위의 과정에서 모두 i좌표와 j좌표에 관한 이중 loop에서 드는 비용이 시간 복잡도가 된다. i좌표와 j좌표 모두 4를 넘지 않아서 상수 번의 loop가 실행되므로 시간 복잡도는 $O(1)$이다. 공간 복잡도는 지역 변수를 선언하지 않고 오직 DrawNextBlock 함수를 호출할 때 드는 비용이므로 $O(1)$이다.

요구사항 ④를 수행하기 위해서는 ProcessCommand 함수 내에서 CheckToMove 함수를 실행해야 한다. ProcessCommand 함수에서 사용자가 입력한 명령에 따라 해당 내용대로 블록을 이동하는 게 가능한지 확인해야 하므로 CheckToMove 함수의 파라미터에 명령대로 이동했을 때의 바뀐 좌표를 넘겨준다. CheckToMove 함수에서는 현재 위치에서 4×4 크기의 칸을 그렸을 때 배열 block의 원소가 차지하는 칸이 모두 필드 내 범위에 속하는지 확인한다. block의 i좌표와 j좌표에 관한 for loop가 시간 복잡도에 영향을 끼치므로 이는 $O(1)$이다. 공간 복잡도는 지역 변수를 선언하지 않고 오직 CheckToMove 함수를 호출할 때 드는 비용이므로 $O(1)$이다.

요구사항 ⑤를 만족하기 위해서는 DrawShadow 함수를 구현해야 한다. 블록이 아래로 내려갈 때 최대한 어느 위치까지 내려갈 수 있는지 알아야 하므로 CheckToMove 함수로 넘겨주는 파라미터의 y좌표를 하나씩 늘려주면서 while loop을 사용하여 반환 값이 0이 될 때까지 변수 i를 하나씩 증가시킨다. loop를 벗어나면 DrawBlock 함수로 넘겨주는 파라미터의 y좌표를 하나 감소시키고 그리는 character를 '/'로 설정하여 이를 수행한다. 현재 블록이 처음에 먼저 그려지고 y좌표가 0일 때부터 높이 전까지 while loop을 수행하므로 시간 복잡도는 최대 $O(HEIGHT)$가 된다. 공간 복잡도는 지역 변수를 선언하지 않고 CheckToMove 함수를 호출할 때 드는 비용이므로 $O(1)$이다.

요구사항 ⑦을 수행하기 위해서는 createRankList 함수를 구현해야 한다. 구체적인 알고리즘은 아래와 같다.

1) rank.txt 파일로부터 랭킹 데이터 수를 입력 받고, 그 수만큼 for loop을 실행하면서 데이터를 하나씩 배열 nodeArray에 입력받는다.
2) sortedBST 함수를 실행하여 nodeArray에 저장된 데이터를 트리 자료구조로 변환한다. 배열에서 중앙에 해당하는 index의 값을 root 노드로 생성하고, 해당 index의 왼쪽에 위치한 배열과 오른쪽에 위치한 배열에 관하여 재귀적으로 sortedBST 함수를 실행한다.
3) 생성된 Tree를 inorder순으로 탐색하면서 랭킹 순위 데이터를 업데이트한다.

랭킹 데이터 수 N만큼 for loop을 사용하여 배열을 생성하고 이를 그대로 트리의 노드로 변환하는 과정에서 드는 비용이 시간 복잡도를 좌우하므로 이는 $O(N)$이다. 공간 복잡도도 N만큼의 배열 원소 개수와 트리에서의 노드 개수이므로 $O(N)$이다.

요구사항 ⑥을 수행하기 위해서는 새로운 데이터를 삽입하기 위해 실행하는 newRank와

insertBST 함수를 구현해야 한다. 데이터를 삽입할 때 점수 값을 기준으로 root 노드의 점수와 비교했을 때 점수가 더 크면 왼쪽 sub tree를, 점수가 더 작거나 같으면 오른쪽 sub tree를 재귀적으로 탐색하면서 노드가 나오지 않을 때까지 최대 트리 깊이만큼 탐색하므로 시간 복잡도는 $O(\log N)$이다. 그리고 순위를 업데이트하는 과정에서 inorder순으로 트리 전체를 탐색하기 때문에 시간 복잡도가 노드 개수에 의존한다. 따라서 시간 복잡도는 $O(N)$이다. 공간 복잡도는 새로 생성된 하나의 노드를 트리에 삽입할 때 드는 비용이므로 노드 하나의 비용인 $O(1)$이다.

요구사항 ⑧ ~ ⑨에서는 rank 함수를 실행하여 switch 문을 통해 명령 값에 따라서 기능을 수행하는데, 명령 값이 1이면 원하는 랭킹 순위 범위에 해당하는 데이터를 출력해야 한다. 첫 번째 순위 값(X)과 두 번째 순위 값(Y) 두 개를 입력 받아 inorder 순으로 트리를 탐색하면서 범위 안에 해당하는 노드의 데이터를 출력한다. 단, 입력 받을 때의 예외 처리가 필요하다. 예외 처리는 아래와 같다.

1) X와 Y 모두 아무 것도 입력받지 않으면 1위와 5위를 inOrder 함수의 순위에 관한 파라미터로 넘긴다.
2) X만 입력받지 않으면 1위와 입력받은 Y를 inOrder 함수의 순위에 관한 파라미터로 넘긴다.
3) Y만 입력받지 않으면 입력받은 X와 X + 4를 inOrder 함수의 순위에 관한 파라미터로 넘긴다.
4) X가 Y보다 크면 리스트에 있는 순위가 아니라는 경고문을 출력한다.

시간 복잡도는 inorder 방식으로 최대 전체 노드를 탐색하므로 $O(N)$이 걸린다.

명령 값이 2이면 원하는 이름에 해당하는 데이터를 출력한다. 여기서도 마찬가지로 inorder 방식으로 찾고자 하는 이름과 같은 데이터를 저장하는 노드를 찾아서 출력한다. inOrder() 함수를 변형한 inOrderSearchName 함수를 통해서 이를 수행하며, strcmp 함수를 사용하여 노드에 저장된 이름과 찾고자 하는 이름이 서로 같은지 확인한다. 시간 복잡도는 inorder 방식으로 최대 전체 노드를 탐색하므로 $O(N)$이 걸린다. 공간 복잡도는 원하는 데이터를 찾을 때는 기존에 생성되어 있는 트리를 참조하고 노드 생성과 같은 작업은 진행하지 않으므로 $O(1)$이다.

위의 rank 함수에서 명령 값을 3으로 입력하면 원하는 데이터 삭제 기능을 수행한다. 삭제 기능은 deleteBST 함수에서 수생하는데, 여기서는 inorder 방식으로 탐색하지 않고 앞에서 기술한 insertBST 함수와 유사하게 재귀적으로 함수를 호출하여 삭제하고자 하는 데이터를 찾는다. 그러나 노드를 삭제하면 BST가 inorder 순으로 탐색했을 때 순위 순서대로 저장되지 않고 흐트러지기 때문에 이를 다시 정렬해줘야 한다. 이에 대한 구체적인 알고리즘은 다음과 같다.

1) 삭제하고자 하는 노드의 왼쪽 자식 노드가 존재하지 않으면 삭제하는 노드 자리에 오른쪽 자식 노드를 채워야 한다. 그래서 노드의 right 포인터를 반환하여 재귀적으로 함수를 호출했을 때 위의 부모 노드가 반환 값을 가리키도록 한다.
2) 삭제하고자 하는 노드의 오른쪽 자식 노드가 존재하지 않으면 1)번과 같은 과정을 수행하여 노드의 left 포인터를 반환해서 왼쪽 자식 노드를 채워야 한다.
3) 왼쪽과 오른쪽 자식 노드가 모두 존재하면 삭제하고자 하는 노드의 오른쪽 노드에서 inorder

순으로 탐색했을 때 가장 왼쪽에 위치한 자식 노드를 찾아서 삭제하는 노드 자리를 채우고 오른쪽 sub tree에 관하여 재귀적으로 함수를 실행한다.

시간 복잡도는 이진 탐색 트리의 최대 깊이만큼 재귀적으로 원하는 데이터를 탐색하는 과정에서 걸리는 비용이므로 $O(\log N)$이다. 그리고 inorder 순으로 전체 트리를 탐색하면서 랭킹 순위를 조정해주는 과정에서 걸리는 시간 복잡도가 $O(N)$이다. 공간 복잡도는 상기한 바와 같이 기존에 생성되어 있는 트리를 참조하고 노드 생성 작업은 진행하지 않으므로 $O(1)$이다.

요구사항 ⑩을 수행하기 위한 구체적인 알고리즘은 아래와 같다.

1) 모든 방향을 고려하는 rBlockR 변수와 x좌표에 관한 rBlockX에 관하여 노드를 생성하고 해당 블록이 최대한 내려갈 수 있는 위치를 CheckToMove 함수를 통해 구한다.
2) AddBlockToField와 DeleteLine 함수를 통해 해당 위치에 놓았을 때 얻을 수 있는 점수를 구한다. 참조하고자 하는 다음 블록의 개수 깊이보다 (Ⓐ)*level이 낮으면* 현재 블록에 해당하는 노드를 root로 설정하여 재귀적으로 함수를 실행했을 때 반환 값을 점수에 더한다.
3) 기존에 기록했던 최대 점수보다 현재 노드에서 구한 누적 점수가 더 크면 이를 최대 점수로 업데이트한다. 만약 부모 노드가 root 노드이면 현재 블록이 자식 노드인 level에 도달했으므로 (Ⓑ)*최대 점수와 y좌표 값을 비교하여 이를 놓일 수 있는 위치로 업데이트*한다.

1)번 과정에서 블록이 최대 어느 위치까지 내려올 수 있는 과정을 고려하고, 일정 깊이를 갖는 노드를 M개 생성하므로 참고하고자 하는 트리의 깊이를 $Depth$라고 할 때 시간 복잡도는 최대 $O(M^{Depth} \times HEIGHT)$이다. 공간 복잡도는 생성하는 트리의 노드 개수만큼의 비용이므로 $O(M^{Depth})$이다.

요구사항 ⑪에서는 ⑩에서의 트리를 최적화하여 시간 및 공간 복잡도를 줄여야 하는데, 이를 위해 트리 가지치기 방법을 사용했다. ⑩의 2)번 과정에서 Ⓐ를 “level − 1”로 바꾸고, Ⓑ를 “누적 점수가 높은 5개의 노드의 정보를 저장하는 maxPath와 maxValue 배열에 점수가 높은 순에서 낮은 순으로 저장한다”로 바꾼다. 이후 5개의 노드에 관하여 다시 fastRecommend 함수를 실행하여 앞의 과정과 동일한 과정을 반복한다. 5개의 노드만 깊이 끝까지 탐색하고 나머지 노드에 관해서는 깊이의 한 level 전까지만 탐색하므로 밑이 상수인 exponential 항은 고려하지 않는다고 할 때 시간 복잡도는 $O((n-5)^{depth-1} \times HEIGHT)$이고, 공간 복잡도는 $O((n-5)^{depth-1})$이 된다.

요구사항 ⑫에서는 블록이 놓일 수 있는 경우에 관하여 가중치를 부여하는 방법을 사용한다. 점수를 얻는 과정에서 가장 중요한 경우는 줄을 삭제하는 것이므로 DeleteLine 함수를 통해 계산한 점수 값에 가중치를 가장 높게 부여한다. 그 이후 중요도가 높은 순으로 양수의 가중치를 부여하는데, 이는 블록을 필드 양쪽 side에 맞닿게 놓는 경우, 다른 블록과 옆면에서도 닿는 경우, 그리고 필드의 맨 밑바닥에 놓는 경우이다. 그러나 블록을 놓았을 때 블록의 개수가 많으면 안 되므로 생성되는 블록의 개수를 세서 음수의 가중치를 부여한다. 경우에 따른 가중치의 상대적인 비중과 각 경우마다 관련된 실행 함수는 [표 2]와 같다.

구멍 가중치	줄 삭제 가중치	양 옆 가중치	맞닿는 가중치	맨 밑바닥 가중치
−1	+20	+1	+2	+2
evaluateWeight	DeleteLine	AddBlockToField	AddBlockToField	AddBlockToField

[표 2] 경우에 따른 가중치와 관련 함수

2.4 제작

각 함수의 구체적인 알고리즘은 2.2와 2.3에서 자세히 서술하였다. [표 3]과 [표 4]를 통해 프로그램 완성에 필요한 전역 변수와 함수들 위주로 역할, 파라미터, 그리고 반환 값을 정리한다.

void DrawNextBlock(int* nextBlock)		**void DrawBlock(int y, int x, int blockId, int blockRoate, char tile)**	
역할	블록 ID와 nextBlock의 데이터를 바탕으로 다음에 나올 블록을 해당 영역에 그린다.	**역할**	필요한 데이터를 입력 파라미터로 전달받아서 각 변수에 맞게 정해진 위치에 블록을 그린다.
파라미터	nextBlock: 다음 블록 정보를 갖는 배열	**파라미터**	y : 블록의 y좌표 x: 블록의 x좌표 blockID: 현재 블록의 형태를 알려주는 데이터 blockRotate: 블록을 어떻게 회전해야 하는지 알려주는 데이터 tile: 블록을 그리기 위한 블록 한 칸
반환 값		**반환 값**	
int CheckToMove(char f[HEIGHT][WIDTH], int currentBlock, int blockRotate, int blockY, int blockX)		**void DrawChange(char f[HEIGHT][WIDTH], int command, int currentBlock,int blockRotate, int blockY, int blockX)**	
역할	Command에 의해서 가능한 움직임이 가능한지 판단한다.	**역할**	명령에 의해 바뀐 부분을 필드에 새로 그린다.
파라미터	f[HEIGHT][WIDTH]: 필드 정보를 갖는 배열 currentBlock: 블록의 ID blockRotate: 블록의 회전 방향 blockY: 블록의 y좌표 blockX: 블록의 x좌표	**파라미터**	f[HEIGHT][WIDTH]: 필드 정보가 담긴 배열 currentBlock: 블록의 ID blockRotate: 블록의 회전 방향 blockY: 블록의 y좌표 blockX: 블록의 x좌표
반환 값	움직일 수 있다: 1 움직일 수 없다: 0	**반환 값**	
void BlockDown(int sig)		**void AddBlockToField(char f[HEIGHT][WIDTH], int currentBlock, int blockRotate, int blockY, int blockX)**	
역할	일정 시간마다 자동으로 실행되는 함수이며, 블록을 정해진 시간마다 아래로 한 칸씩 내린다. 만일 블록을 더 이상 내릴 수 없으면 블록을 쌓는 기능을 수행한다. 한 줄이 꽉 채워졌을 때는 줄을 없애는 기능을 수행해야 하면서 score도 계산해야 한다. 더 이상 테트리스를 진행할 수 없으면 게임을 종료한다.	**역할**	BlockDown 함수에서 CheckToMove 함수 호출을 통해 블록을 더 이상 움직일 수는 없지만 기존 필드에 쌓을 수 있을 때 현재 블록을 쌓는다.
파라미터	sig: 매초마다 실행하기 위한 신호	**파라미터**	f[HEIGHT][WIDTH]: 필드 정보를 갖는 배열 currentBlock: 블록의 ID blockRotate: 블록의 회전 방향 blockY: 블록의 y좌표 blockX: 블록의 x좌표
반환 값		**반환 값**	
int DeleteLine(char f[HEIGHT][WIDTH])		**void createRankList()**	
역할	지울 수 있는 줄을 찾아서 지우고 점수를	**역할**	파일로부터 데이터를 읽어서 랭킹 데이터를

	업데이트한다.		노드로 갖는 이진 탐색 트리를 생성한다.
파라미터	f[HEIGHT][WIDTH]: 필드 정보를 갖는 배열	**파라미터**	
반환 값	줄을 삭제했을 때 얻는 점수	**반환 값**	
void DrawShadow(int y, int x, int blockID, blockRotate)		**void DrawBlockWithFeatures(int y, int x, int blockID, int blockRotate)**	
역할	현재 블록의 그림자를 그린다.	**역할**	추천 위치와 현재 블록을 그리며, recommend play를 실행하지 않으면 그림자를 그린다.
파라미터	y: 블록의 y좌표 x: 블록의 x좌표 blockID: 블록의 ID blockRotate: 블록의 회전방향	**파라미터**	y: 블록의 y좌표 x: 블록의 x좌표 blockID: 블록의 ID blockRotate: 블록의 회전방향
반환 값		**반환 값**	
void rank()		**void newRank(int score)**	
역할	입력한 명령에 따라서 원하는 랭킹 데이터를 검색하여 출력하거나 삭제한다.	**역할**	GameOver가 되었을 때 랭킹 정보를 새로 생성하여 트리에 삽입하고 파일에 저장한다.
파라미터		**파라미터**	score: 얻은 점수
반환 값		**반환 값**	
Node* newNode(char* nameData, int scoreData, int nodeRank)		**Node* sortedBST(Node* arrayData, int start, int end)**	
역할	이진 탐색 트리에 저장할 랭킹 정보 노드를 생성한다.	**역할**	랭킹 정보가 저장된 배열을 이진 탐색 트리로 변환한다.
파라미터	nameData: 저장할 이름 scoreData: 저장할 점수 nodeRank: 저장할 순위	**파라미터**	arrayData: 랭킹 정보가 저장된 배열 start: 배열의 첫 index end: 배열의 끝 index
반환 값	생성한 노드	**반환 값**	변환한 노드
Node* insertBST(Node* root, char* nameData, int scoreData)		**Node* minValueInSubBST(Node* node)**	
역할	이진 탐색 트리에 노드를 삽입한다.	**역할**	inorder순으로 탐색할 때 현재 자신의 노드 기준으로 그 다음에 오는 노드를 구한다.
파라미터	root: 이진 탐색 트리의 root 노드 nameData: 이진 탐색 트리에 삽입할 이름 scoreData: 이진 탐색 트리에 삽입할 점수	**파라미터**	node: 기준이 되는 노드의 오른쪽 자식 노드
반환 값	root 노드	**반환 값**	inorder successor에 해당하는 노드
void inOrderUpdateRank(Node* node, int *updateRank)		**Node* deleteBST(Node* root, int rankData, int* deleteFlag)**	
역할	inorder순으로 트리를 탐색하면서 각 노드의 랭킹 순위를 업데이트한다.	**역할**	삭제하고자 하는 순위에 해당하는 노드를 트리에서 삭제한다.
파라미터	node: 탐색하고자 하는 노드 updateRank, 바꾸고자 하는 순위	**파라미터**	root: 이진 탐색 트리의 root 노드 rankData: 삭제하고자 하는 순위 deleteFlag: 삭제했는지 여부를 저장하는 플래그
반환 값		**반환 값**	root 노드
void inOrder(Node* node, int *counter, int X, int Y)		**void inOrderSearchName(Node* node, char* findName, int* findFlag)**	
역할	inorder순으로 트리를 탐색하면서 입력한 순위 범위에 해당하는 데이터를 출력한다.	**역할**	inorder순으로 트리를 탐색하면서 입력한 이름에 해당하는 데이터를 출력한다.
파라미터	node: 탐색하고자 하는 기준 노드 counter: 현재 노드의 순위 X: 시작 범위의 순위 Y: 끝 범위의 순위	**파라미터**	node: 탐색하고자 하는 기준 노드 findName: 탐색하고자 하는 이름 findFlag: 찾고자 하는 노드를 찾았는지 저장하는 플래그
반환 값		**반환 값**	

int makeRecommendTree()		int recommend(RecNode* root)	
역할	추천 시스템을 사용하기 위해 필요한 트리를 생성한다.	역할	블록의 정보를 참고하여 놓을 수 있는 위치에서 가장 높은 점수를 얻을 수 있는 경우를 찾는다.
파라미터		파라미터	root: 탐색하고자 하는 root 노드 (탐색하고자 하는 블록이 놓이기 전 이전 블록까지 놓였을 때의 상태)
반환 값	트리 생성이 잘 되었다: 0	반환 값	누적 최대 점수 값
int modified_recommend(RecNode* root)		int fastRecommend(RecNode* root)	
역할	가지치기와 가중치 부여를 사용해서 시간 및 공간 복잡도를 줄인 recommend 함수이다.	역할	가자치기를 모두 하고 나서 점수가 높은 상위 5개 노드만 깊이 끝까지 탐색하는 recommend 함수이다.
파라미터	root: 탐색하고자 하는 root 노드 (탐색하고자 하는 블록이 놓이기 전 이전 블록까지 놓였을 때의 상태)	파라미터	root: 탐색하고자 하는 root 노드 (탐색하고자 하는 블록이 놓이기 전 이전 블록까지 놓였을 때의 상태)
반환 값	누적 최대 점수 값	반환 값	누적 최대 점수 값
void recommendedPlay()		int evaluateWeight(char field[HEIGHT][WIDTH])	
역할	메뉴에서 3번 recommend를 입력했을 때 추천 시스템에 따라서 테트리스 게임을 실행한다.	역할	블록을 놓았을 때 몇 개의 구멍이 생기는지 그 수를 세어 음수의 가중치를 부여한다.
파라미터		파라미터	field[HEIGHT][WIDTH]: 필드 정보를 갖는 배열
반환 값		반환 값	구멍에 의한 가중치

[표 3] 프로그램 완성에 필요한 각 함수의 역할, 파라미터, 반환 값 정리

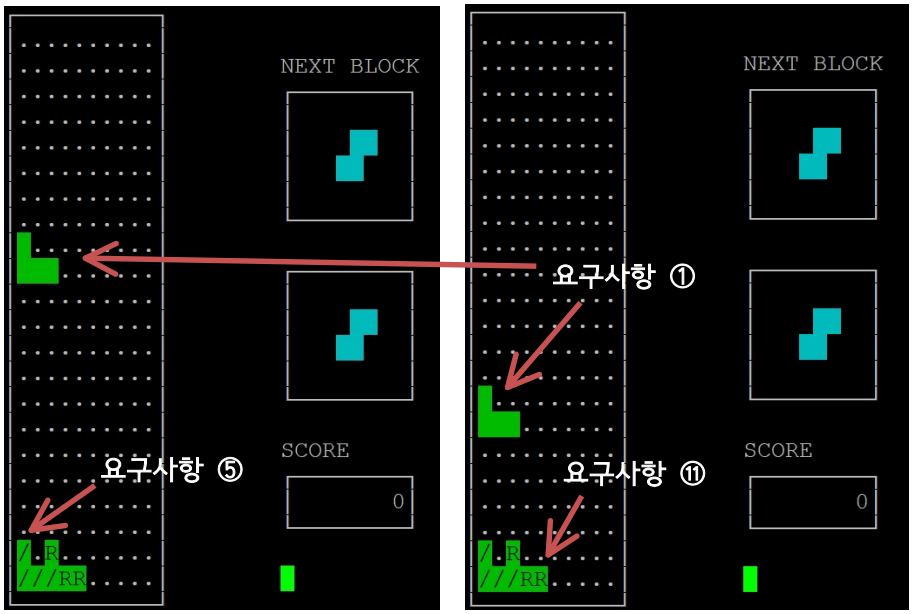
전역 변수명	의미 · 역할	전역 변수명		의미 · 역할
BLOCK_NUM	현재 블록을 포함하여 앞으로 봐야 할 블록의 개수이다.	VISIBLE_BLOCKS		추천 시스템 트리에서 현재 블록을 포함하여 앞으로 참고해야 할 다음 블록의 개수이다.
MAX_BLOCKS	추천 시스템에서 트리 가지치기를 수행할 때 남기려고 하는 가장 높은 누적 점수의 노드의 개수이다.	Node		랭킹 시스템에서 데이터를 저장하는 이진 탐색 트리의 노드의 구조체 data type이다.
RecNode	추천 시스템 트리 노드의 구조체 data type이다.	blockColor[]		블록의 색에 관한 데이터를 갖는 배열이며 원소에 따라 빨강, 초록, 노랑, 파랑, 마젠타, 시안, 하양색을 나타낸다.
fallingBlock	현재 블록인지 아닌지를 나타내는 플래그이다.	dataNum		랭킹 데이터의 총 개수를 의미한다.
recommendPlaying	3번 recommend play를 실행 중인지를 나타내는 플래그이다. 누적 점수를 계산할 때 트리에서의 탐색의 경우를 고려하여 가중치를 고려해야 하는지 아닌지를 알아야 하기 때문이다.	recommendTreeSearchFlag		추천 시스템 트리에서 누적 점수를 구할 때 사전에 설정한 가중치까지 고려한 점수를 구하기 위해서 만든 플래그이다. 이 값이 1일 때는 AddBlockToField 함수에서 점수 대신 가중치를 계산한다.
start	3번 recommend play 기능의 실행 시간을 잴 때 시작 시간이다.	stop		3번 recommend play 기능의 실행 시간을 잴 때 QUIT 또는 Game Over된 시점의 시간, 즉 끝나는 시간이다.
duration	3번 recommend play 기능을 실행할 때 처음부터 게임이 끝날 때까지 걸린 시간이다.	HOLE DELETE SIDE TOUCHED BOTTOM	WEIGHT	최적화된 추천 시스템을 사용할 때 각 경우마다 계산하는 가중치의 값을 갖는다.

maxValue[]	추천 시스템의 트리에서 보고자 하는 깊이 이전 단계까지 누적 점수가 높은 5개의 노드의 누적 점수 값을 저장한다.	maxPath[]	추천 시스템의 트리에서 보고자 하는 깊이 이전 단계까지 누적 점수가 높은 5개의 노드의 놓는 위치 회전 정보를 저장한다.

[표 4] 프로그램 완성에 필요한 각 전역 변수의 이름과 의미 · 역할 정리

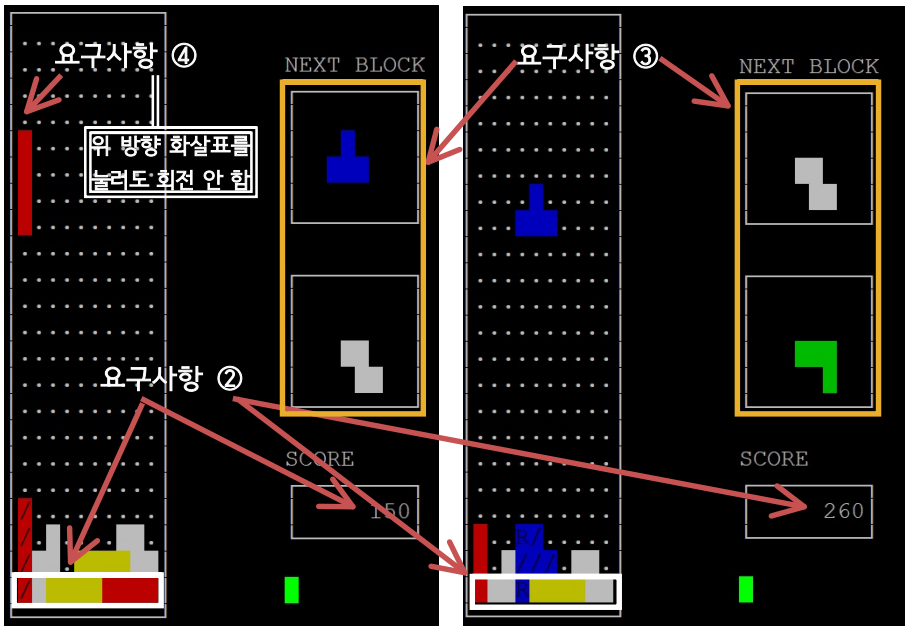
2.5 시험

1) 1번 Play 기능을 수행했을 때



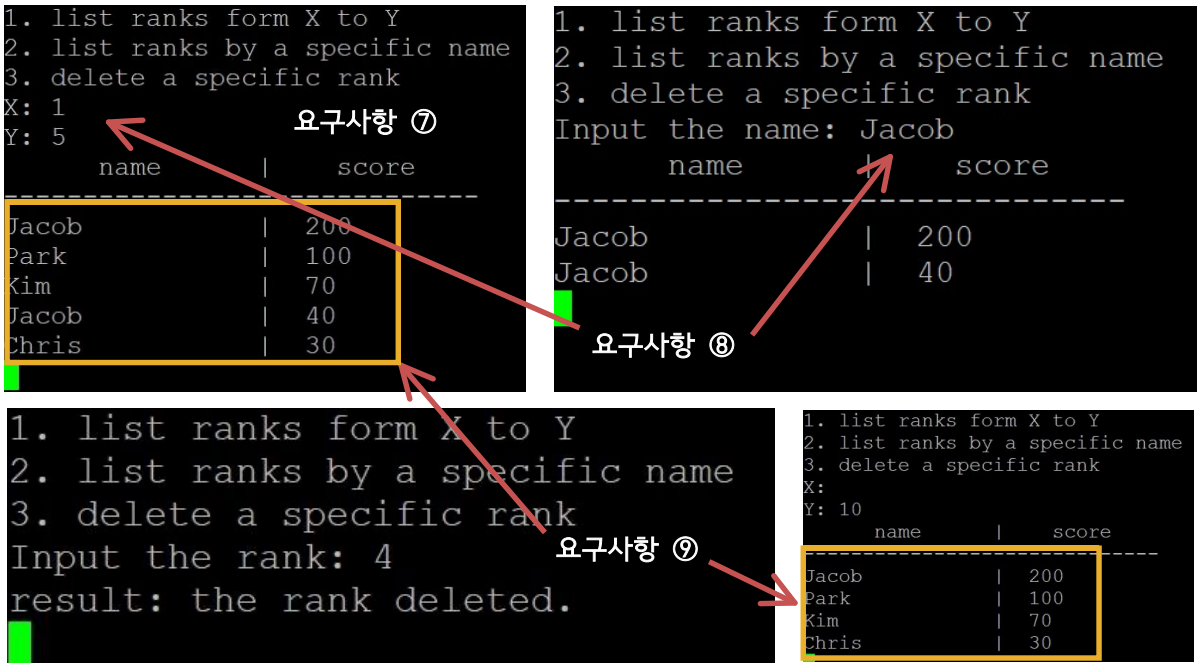
[그림 5]

[그림 5]의 오른쪽 화면은 왼쪽 화면에서 6초가 경과되었을 때의 게임 화면이다. 매초마다 한 칸씩 내려왔으며 현재 블록이 최대한 내려올 수 있을 때의 위치와 추천 위치를 적절히 표시한다는 사실을 알 수 있다.



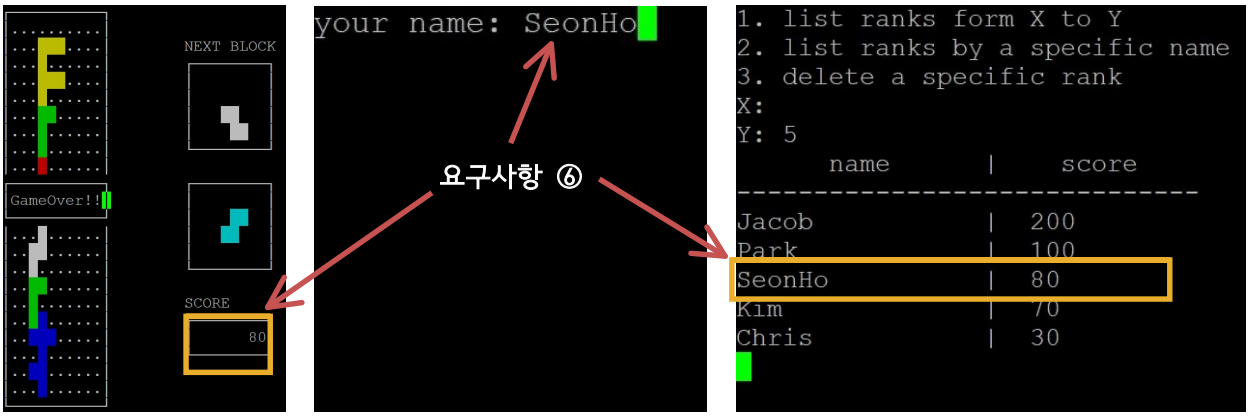
[그림 6]

[그림 6]에서는 블록을 필드에 놓았을 때 아래로 닿는 면적에 따라 점수를 증가시키고 줄이 모두 채워졌을 때 줄을 지우고 추가로 점수를 증가시키는 것을 알려준다. 또한 블록이 필드의 양쪽 가장자리 바로 옆에 위치할 경우 위 방향(↑) 화살표를 누르면 회전이 되지 않는다. 그리고 점수를 얻고 나서 다음 블록의 정보를 출력하는 NEXT BLOCK 영역의 내용이 변경된 사실을 알 수 있다.



[그림 7]

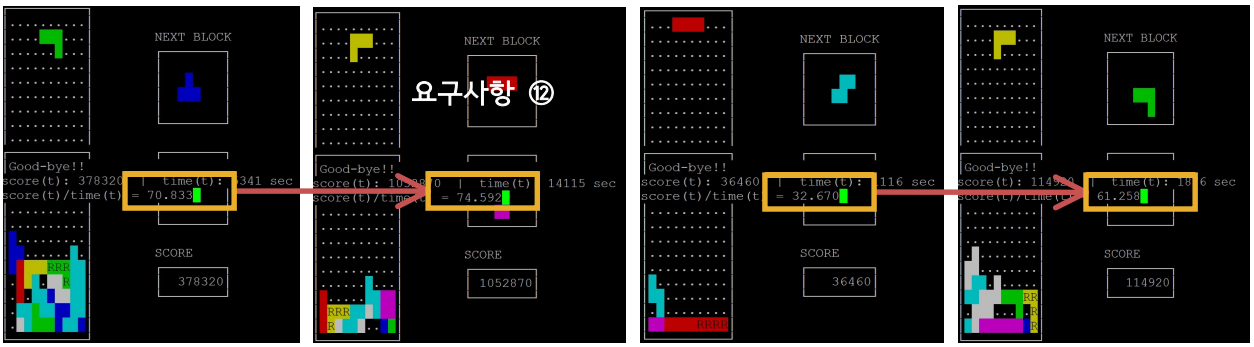
[그림 7]에서 첫 번째 화면은 rank 기능에서 1번을 누르고 순위를 두 개 입력했을 때 그 범위에 해당하는 랭킹 데이터를 출력하는 것을 나타낸다. 두 번째 화면은 2번을 누르고 이름을 입력했을 때 해당 이름을 갖고 있는 랭킹 데이터를 출력하는 것이다. 세 번째 화면은 3번을 누르고 삭제하고자 하는 순위를 입력했을 때 해당 순위를 랭킹 데이터에서 삭제하는 것을 보여준다.



[그림 8]

[그림 8]에서 첫 번째 화면은 필드에 쌓인 블록의 높이가 전체 필드 높이를 초과하여 Game Over가 발생한 상황이다. 두 번째 화면은 Game Over가 뜨고 나서 임의의 키보드 입력을 넣었을 때 Game Over가 뜰 때까지 얻은 점수를 랭킹 데이터에 쓰기 위해 이름도 같이 입

력받는 것을 나타낸다. 세 번째 화면은 앞에서 새로운 점수가 실제로 랭킹 데이터에 삽입되었는지를 확인하기 위해 rank 기능에서 순위를 입력하여 랭킹 데이터를 출력한 것이다.



[그림 9]

[그림 9]에서 (1)번과 (2)번 화면은 참고할 다음 블록의 개수인 VISIBLE_BLOCKS의 값을 3으로 정하고 각각 recommend 함수와 modified_recommend 함수를 임의의 시간동안 실행하고 Quit했을 때 점수 대 시간의 비율을 출력한 것이다. 트리 가지치기와 가중치 부여 방법을 사용하여 추천 시스템을 최적화했을 때의 비율이 74.592로 하지 않았을 때의 비율인 70.833에 비해 상대적으로 더 높게 나온다는 사실을 알 수 있다. 마찬가지로 (3)번과 (4)번 화면은 VISIBLE_BLOCKS의 값을 4로 정하고 recommend 함수와 modified_recommend 함수를 각각 실행했을 때의 비율을 나타낸다. 전자는 32.670인 반면 후자는 이의 두 배에 가까운 61.258이다. 따라서 recommend 함수를 최적화하는 요구사항 ⑫를 만족했다는 것을 알 수 있으며, VISIBLE_BLOCKS의 값을 높일수록 점수 대 시간의 비율은 최적화를 안했을 때와 할 때의 차이가 더 커진다는 사실을 알 수 있다.

2.6 평가

위의 내용을 종합하면 이번 프로젝트의 요구사항 ① ~ ⑫를 적절히 수행했다는 사실을 알 수 있다. 위에 명시된 요구사항들 외의 프로젝트에서 세세하게 요구하는 사항들도 오류 없이 수행된다. tetris.h에 정의된 BLOCK_NUM 값을 2 이상으로 변경해도 다음 블록 정보가 정확히 표시되며, 1번 play에서 요구되는 기본적인 테트리스의 구성 요소와 기능이 문제없이 실행됨을 확인했다.

2번 rank에서 요구되는 사항들도 모두 만족한다. 특히, N개의 특정 데이터를 삽입, 검색, 또는 삭제하는 데 있어서 linked list의 최대 시간 복잡도인 $O(N)$보다 더 좋은 시간 복잡도인 $O(\log N)$을 갖는 binary search tree를 사용했다. 그러나 삽입 또는 삭제를 수행하면 트리에 저장된 데이터의 순위를 다시 정렬해주는 작업을 수행하는 데 시간 복잡도가 최대 $O(N)$이 소요되기 때문에 데이터를 검색하는 것 외에 새로운 데이터를 삽입하거나 기존 데이터를 삭제하는 데 있어서 최종적으로는 시간 복잡도 면에서 이득을 얻었다고 보기는 어렵다.

3번 recommend와 추천 시스템 제작에 요구되는 사항들도 빠짐없이 만족한다. 추천 시스템 트리에서 앞으로 봐야 할 다음 블록의 개수인 VISUAL_BLOCKS 값을 늘려도 정상적으로 추천 위치를 그린다. 또한 '2.5 시험'에서의 결과를 고려했을 때 추천 시스템의 점수 효율을 높이기 위해서 사용한 트리 가지치기와 가중치 부여도 성공적이다. 특히, VISUAL_BLOCKS의 값을 증가시킬수록 점수 대 시간의 비율은 최적화를 하지 않을 때보다 상대적으로 더 높다.

2.7 환경

학생들이 리눅스 서버를 접속하여 프로젝트를 진행하므로 해당 서버에 접속할 수 있는 데스크탑과 ssh 접속 프로그램을 제공한다. 접속하는 리눅스 서버에 각 학생들에게 하위 계정을 발급하여 할당받는 용량에 한하여 자유롭게 이를 이용하여 프로젝트를 진행할 수 있는 환경을 제공한다.

2.8 미학

1주차에서 3주차 실습을 모두 진행하면서 모두 같은 기존 인터페이스 내에서 기능을 추가하거나 수정하여 최초로 주어지는 프로그램의 인터페이스가 일관적으로 유지된다.

새롭게 추가한 것은 테트리스 블록에 색이 칠해지는 것이다. 테트리스의 7개 블록 별로 각각 빨강, 초록, 노랑, 파랑, 마젠타, 시안, 하양 7개의 색을 배열에 저장했다. 블록을 그릴 때마다 DrawBlock 함수에서 attron 또는 attroff 함수를 수행할 때 블록의 색 배열 원소를 인자로 넘긴다. 이 함수들은 글자에 특수한 효과를 주는 함수이며 '|(OR)'를 써서 한 번에 여러 인자를 넘겨서 다양한 효과를 줄 수 있다. 그래서 COLOR_PAIR 매크로를 사용하여 반환한 값을 앞의 함수들에 파라미터로 넘겼다.

2.9 보건 및 안정

이 프로그램에서 오류가 발생하여 프로그램이 무한 루프에 빠지거나 강제 종료가 되는 현상을 막기 위해 예외 처리를 몇 가지 진행했다.

랭킹 시스템에서 읽어 올 파일이 없을 경우 자동으로 파일을 생성하고 사용자가 play 또는 recommend play 기능을 실행하여 새롭게 얻은 점수를 데이터로 저장할 수 있도록 했다. 그러나 이 파일에 데이터의 개수는 기록하지 않기 때문에 프로그램을 종료하고 다시 파일을 읽어올 경우 오류가 발생할 수 있다.

메모리가 동적으로 할당된 것들에 관해 프로그램이 종료되었을 때 메모리에 그대로 남아있지 않도록 free 함수를 통해 동적 할당을 해제했다.

시간 복잡도를 최적화한 추천 시스템의 트리에서 봐야할 깊이가 2보다 작거나 같은 경우 기존 추천 시스템 트리를 사용하는 것과 효율적인 면에서 거의 차이가 발생하지 않을 뿐더러 봐야할 깊이의 이전 깊이가 0이 되어 오류가 발생할 수 있어서 modified_recommend 함수 대신 기존 추천 시스템 트리를 탐색하는 recommend 함수를 수행한다.

3. 기 타

3.1 환경 구성

ncurses library는 윈도우 패널, 메뉴, 마우스, 색상 등을 쉽게 사용할 수 있도록 도와주는 라이브러리이다. 이 라이브러리의 사용을 위해서는 #include <ncurses.h>를 헤더 파일 또는 소스에 추가하고, 컴파일 시 -lncurses 옵션을 줘야 한다. 이 프로그램에서 사용한 ncurses library 함수는 echo, noech, refresh, clear, prinw, scanw 등이 있다. [표 5]는 ncurses library의 각 함수의 기능을 나타낸 것이다.

echo noecho	사용자로부터 입력받은 문자를 출력할지 그 여부를 결정하는 함수이다
refresh	화면에 표시할 내용을 갱신하는 함수이다.
clear	화면을 모두 지워주는 함수이다.
printw	입출력을 담당하는 함수이다.

[표 5] ncurses library의 각 함수의 기능

vi(vim)는 유닉스 환경에서 가장 많이 쓰이는 문서 편집기이며, 환경 설정을 통해 C언어 소스코드의 자동 줄맞춤, 컬러 코드 지정 등의 기능을 이용할 수 있다. vi는 버전 7.4.52의 Vi IMproved를 사용하였다.

gcc는 GNU 컴파일러 모음의 약자로, GNU 프로젝트의 일환으로 개발되어 널리 쓰이고 있는 컴파일러이다. UNIX 계열 시스템에서 C언어로 프로그램을 개발할 수 있도록 전처리기, 컴파일러, 어셈블러, 링커 등의 역할을 수행한다. gcc는 버전 Ubuntu 4.8.2-19ubuntu1을 사용하였다. Ubuntu는 cspro 서버가 작동하는 리눅스 운영 체제이다. cspro는 서강대학교 컴퓨터공학과에서 운영하는 서버이며, 위의 모든 기능들이 cspro 서버에 설치되어 있다.

3.2 참고 사항

3주차 실습의 결과물로 제출한 코드에서 3주차 과제로 제출한 코드로 바꾸는 과정에서 테트리스 각 블록마다 색을 지정하고 이를 출력하기 위해 적지 않은 변화가 있었다. 프로그램의 코드를 확인할 때 후자를 참고하는 것을 추천한다.

3.3 팀 구성

20171665학번 이선호 수강생이 1명이 100% 참여했다.

3.4 수행기간

2018년 11월 9일부터 2018년 12월 7일까지 수행했다.