CSE3013 컴퓨터공학 실험: 테트리스 프로젝트

담당교수: 서강대학교 컴퓨터공학과 김승욱

분반 : 1반 학번 : 20160641 이름 : 조보현

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. 1주차부터 3주차까지의 설계 문제 및 목표는 다음과 같다.

1) 1주차

1주차의 목표는 제공된 프레임(frame)프로그램을 바탕으로, 블록의 이동, 회전, 필드에 쌓기, 줄 삭제, 점수 계산, 블록 미리보여주기 등 기본적인 기능을 갖는 테트리스 게임을 구현하고, ncurses 라이브러리, 디버깅 방법, makefile을 만드는 방법등을 익히는 것이다. play() 함수 및 blockDown()함수를 구현하는 것이 게임을 구현하는데 있어 가장 중요한 내용이다. 1주차 실습 및 과제를 수행함으로써 달성해야할 설계 문제는 다음과 같다.

① 일정한 시간마다 블록이 아래로 떨어지며, 사용자로부터 입력받은 키에 따라 블록이 알맞은 방향으로 이동해야 한다.

② 블록이 떨어져 다른 블록에 닿거나 바닥에 닿는 경우 면적에 따라 점수를 추가한다.

③ 한 줄의 블록이 채워질 경우 해당 줄을 삭제하고 삭제된 줄 수에 비례하여 점수를 추가하며, 사라진 줄 수만큼 다른 블록들이 아래로 이동해야 한다.

④ q/Q키를 입력할 경우 프로그램을 종료한다. 혹은 블록이 더 이상 필드에 쌓일 수 없는 경우 게임종료 문구를 화면에 띄우고 게임을 종료한다.

⑤ 블록이 떨어질 위치가 그림자 기능을 통해 표현된다.

⑥ 다음 블록을 화면상에서 나타낸다.

2) 2주차

2주차의 목표는 테트리스의 랭킹 시스템을 구현하는 것이다. 테트리스 프로그램 내에서 효율적인 자료구조를 사용하여 랭킹을 관리하고, rank.txt 파일을 이용해 지속적으로 유지한다. 2주차 실습 및 과제를 수행함으로써 달성해야할 설계 문제는 다음과 같다.

① 사용자가 테트리스를 중도에 중단시킨경우가 아닌, 정상적으로 플레이후 종료될 시 이름을 입력 받아 랭킹에 추가한다.

② rank.txt에 담긴 랭킹 정보를 불러와 자료구조에 저장시킨다.

③ 모든 랭킹 정보는 rank.txt에 유지되게 한다. 프로그램이 실행도중 중단되어도 랭킹 정보는 유지되도록 한다.

④ 사용자로부터 입력받은 특정 범위의 랭킹을 출력, 특정 이름의 랭킹 출력, 특정 랭킹의 삭제 기능을 구현한다.

3) 3주차

3주차의 목표는 사용자를 위해 어떤 위치에 블록을 놓으면 높은 점수를 받을 t 있을지 추천하는 추천 시스템을 구현하고, 이를 자동으로 플레이하는 기능을 구현하는 것이다. 이를 통해 트리(Tree)구조와 포인터에 대한 이해를 높일 수 있도록 한다. 3주차 실습 및 과제를 수행함으로서 달성해야할 설계 문제는 다음과 같다.

① 고려할 블록이 필드에 쌓일 수 있는 모든 경우의 수를 고려해 가장 높은 점수를 얻을 수 있는 위치를 그림자 기능을 통해 표현한다.

② 자동으로 플레이할 경우 추천되는 위치로 블록이 바로 옮겨지도록 한다.

③ 모든 경우의 수를 고려하는 것보다 효율적인 방법을 구상해 구현하도록 한다.

1.1 현실적 제한조건

이 테트리스 게임은 문자(character)로 구현되어 있기 때문에 정확한 블록이나 필드의 표현이 어렵다. 특히 인코딩에 따라서 그래픽이 깨져 보이기도 하며 사용환경에 따라서 플레이가 어려운 경우들이 있다. 예를 들어 ncurses 라이브러리가 없는 환경이라면 이 프로그램을 컴파일하고 실행하기 힘들다. 이론적으로 이 프로그램은 사용자에게 가장 높은 점수를 얻을 수 있는 위치를 추천해주어야 하지만, 이는 현실적으로 구현하기가 쉽지 않다. 모든 경우의 수를 고려하는 알고리즘을 사용하면 이론적으로 최고의 점수를 얻을 수 있겠지만, 현실적으로는 메모리와 컴퓨터의 성능 등으로 인해 조금만 데이터가 많아져도 프로그램의 진행이 어렵다.

2. 요구사항

2.1 설계 목표 설정

	목표	요구사항
1주차	①일정한 시간마다 블록이 아래로 떨어지며, 사용자로부터 입력받은 키에 따라 블록이 알맞은 방향으로 이동해야 한다.	일정한 시간마다 블록이 아래로 떨어진다.
		위쪽 키를 누를 시 블록이 회전한다.
		오른쪽, 아래, 왼쪽 키를 누를 시 해당방향으로 한 칸 블록이 이동한다.
	② 블록이 떨어져 다른 블록에 닿거나 바닥에 닿는 경우 면적에 따라 점수를 추가한다.	바닥에 닿거나 다른 블록에 닿은 면적*10 만큼 score가 상승한다,
	③ 한 줄의 블록이 채워질 경우 해당 줄을 삭제하고 삭제된 줄 수에 비례하여 점수를 추가하며, 사라진 줄 수만큼 다른 블록들이 아래로 이동해야 한다.	한 줄의 블록이 모두 채워지면 해당 줄이 삭제된다.
		(삭제 된 줄 수)^2 * 100 만큼 score가 상승한다.
		삭제 된 줄 수만큼 다른 블록들이 아래 이동한다.
	④ q/Q키를 입력할 경우 프로그램을 종료한다. 혹은 블록이 더 이상 필드에 쌓일 수 없는 경우 게임종료 문구를 화면에 띄우고 게임을 종료한다.	q/Q키를 입력할 경우 "Good-bye!!"라는 문구를 출력하고 게임을 종료한다.
		블록이 더 이상 쌓일 수 없는 경우를 블록이 이동할 때 마다 판단한다.
		블록이 더 이상 쌓일 수 없는 경우

		"Game-Over!!"라는 문구를 출력하고 게임을 종료한다.
	⑤ 블록이 떨어질 위치가 그림자 기능을 통해 표현된다.	블록이 떨어질 위치를 '/' 글자로 필드상에 표현한다.
	⑥ 다음 블록을 화면상에서 나타낸다.	nextBlock 에 다음으로 나타날 블록을 그려서 표현한다.
2주차	① 사용자가 테트리스를 중도에 중단시킨 경우가 아닌, 정상적으로 플레이후 종료될 시 이름을 입력 받아 랭킹에 추가한다.	게임이 종료 될 시 사용자로부터 이름을 입력받는다.
		입력받은 이름과 점수를 토대로 랭킹에 정보를 추가한다.
	② rank.txt에 담긴 랭킹 정보를 불러와 자료구조에 저장시킨다.	rank.txt에 담긴 랭킹정보를 효율적인 자료구조에 저장한다.
		저장할 때는 score를 기준으로 내림차순으로 저장한다.
		rank.txt 파일이 존재하지 않더라도 오류가 나지 않도록 구현한다.
	③ 모든 랭킹 정보는 rank.txt에 유지되게 한다. 프로그램이 실행도중 중단되어도 랭킹 정보는 유지되도록 한다.	프로그램이 실행도중 중단되어도 rank.txt파일이 유지되도록 한다.
		rank.txt에 저장될 때에도 score를 기준으로 내림차순으로 저장한다.
	④ 사용자로부터 입력받은 특정 범위의 랭킹을 출력, 특정 이름의 랭킹 출력, 특정 랭킹의 삭제 기능을 구현한다.	X부터 Y까지 범위의 score를 가진 사람들의 정보를 출력한다.
		X와 Y가 설정되지 않는 경우 모든 랭킹을 출력한다.
		X와 Y가 정상적인 범위가 아닐 경우 오류메시지를 출력한다.
		X는 항상 Y보다 작거나 같아야 하며 Y가 설정되지 않으면 범위가 끝까지 인 것으로 추정한다.
		특정 이름의 랭킹을 출력한다. 동명이인의 정보도 출력한다.
		특정 랭킹은 랭킹 넘버를 기준으로 삭제한다. (ex: 3을 입력할 경우 세 번째 랭킹 삭제)
3주차	① 고려할 블록이 필드에 쌓일 수 있는	모든 경우의 수를 고려해 가장 높은

	모든 경우의 수를 고려해 가장 높은 점수를 얻을 수 있는 위치를 그림자 기능을 통해 표현한다.	점수를 얻을 수 있는 위치를 표현한다.
		그림자 기능을 통해 추천 위치를 'R'타일로 표현한다.
		고려할 블록의 수를 사용자가 결정하도록 한다.
	② 자동으로 플레이할 경우 추천되는 위치로 블록이 바로 옮겨지도록 한다.	사용자는 q/Q 입력 외의 아무 입력도 하지 않는다. q/Q입력시 프로그램이 종료된다.
		프로그램이 자동으로 추천 위치로 블록을 이동시키고 게임을 플레이 한다.
	③ 모든 경우의 수를 고려하는 것보다 효율적인 방법을 구상해 구현하도록 한다.	효율적인 자료구조를 구상한다.
		효율적인 알고리즘을 구상한다.

2.2 합성

1) 1주차 자료구조/알고리즘

1주차에서는 테트리스의 기본기능들을 구현한다. 이때 필드와 블록을 표현하기 위해서 사용할 자료구조는 2차원 배열이다. 2차원 배열을 사용하면 x,y좌표를 나타내기 쉬우므로 적절하다. 필드는 HEIGHT*WIDTH 크기의 2차원 배열, 블록은 4*4 크기의 2차원 배열로 표현한다.

2) 2주차 자료구조/알고리즘

2주차에서는 테트리스의 랭킹 기능을 구현한다. 이 때 랭킹 정보를 저장하기 위한 자료구조는 연결리스트이다. 각 노드에 랭킹 정보들을 저장하고, 한 방향으로 연결되어 있는 단방향 연결리스트 구조를 사용한다. 연결리스트를 사용하면 내림차순으로 정보를 저장하고, 삽입, 삭제 및 탐색할 때 용이하다. 각 노드의 구성은 다음과 같다.

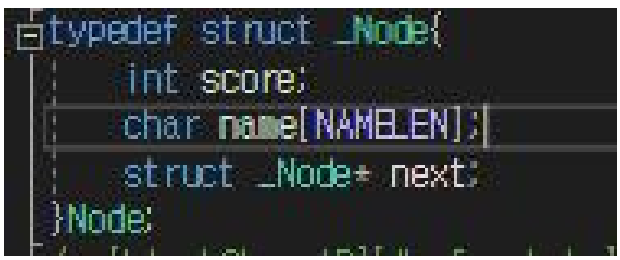

그림 1 랭킹 시스템 노드 구성

점수를 기록하는 int score 변수와 이름을 저장하는 char name[], 그리고 다음 노드를 가리키는 포인터 _Node* next 로 노드가 구성되어 있다. 또한 전역변수로 연

결리스트의 헤더노드를 가리키는 포인터 Node* head를 선언해준다. 이를 통해 연결리스트의 탐색, 삽입, 삭제등을 구현한다.

3) 3주차 자료구조/알고리즘

3주차에서는 테트리스의 추천기능을 구현한다. 이 때 추천기능을 구현하기 위해서 사용할 자료구조는 트리이며 알고리즘은 Brute Force 방식이다. 모든 경우의 수를 표현하기 위해서 복잡한 관계를 트리구조로 표현하면 상대적으로 간단하게 나타낼 수 있으며, 모든 경우의 수를 고려하기 위해서는 Brute Force 알고리즘을 사용해야 한다. 더 향상된 성능의 자료구조를 구현하기 위해서는 AVL트리 혹은 레드-블랙 트리를 구상할 수 있다. 더 향상된 알고리즘을 구현하기 위해서는 특정 경우는 고려하지 않는 가지치기 방식 pruning 알고리즘을 구상한다. 이러한 기능을 구현하기 위한 트리구조의 각각의 노드들의 구성은 다음과 같다.

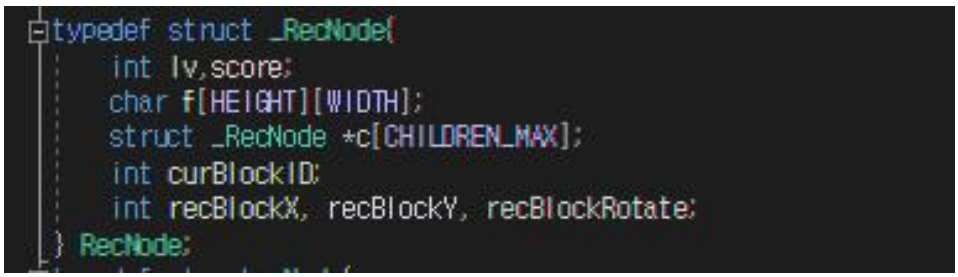

그림 2 추천기능 트리 구조의 노드 구성

트리의 깊이를 나타내는 int lv, 누적 점수를 나타내는 int score, 필드 정보를 기록하는 char f[][], 자식 노드들을 가리키는 포인터 _RecNode *c[]. 블록 ID를 기록하는 int curBlockID, 추천 블록의 좌표 및 회전 정보를 기록하는 int recBlockX, BlockY, recBlockRotate 로 구성되어 있다.

2.3 분석

1) 1주차

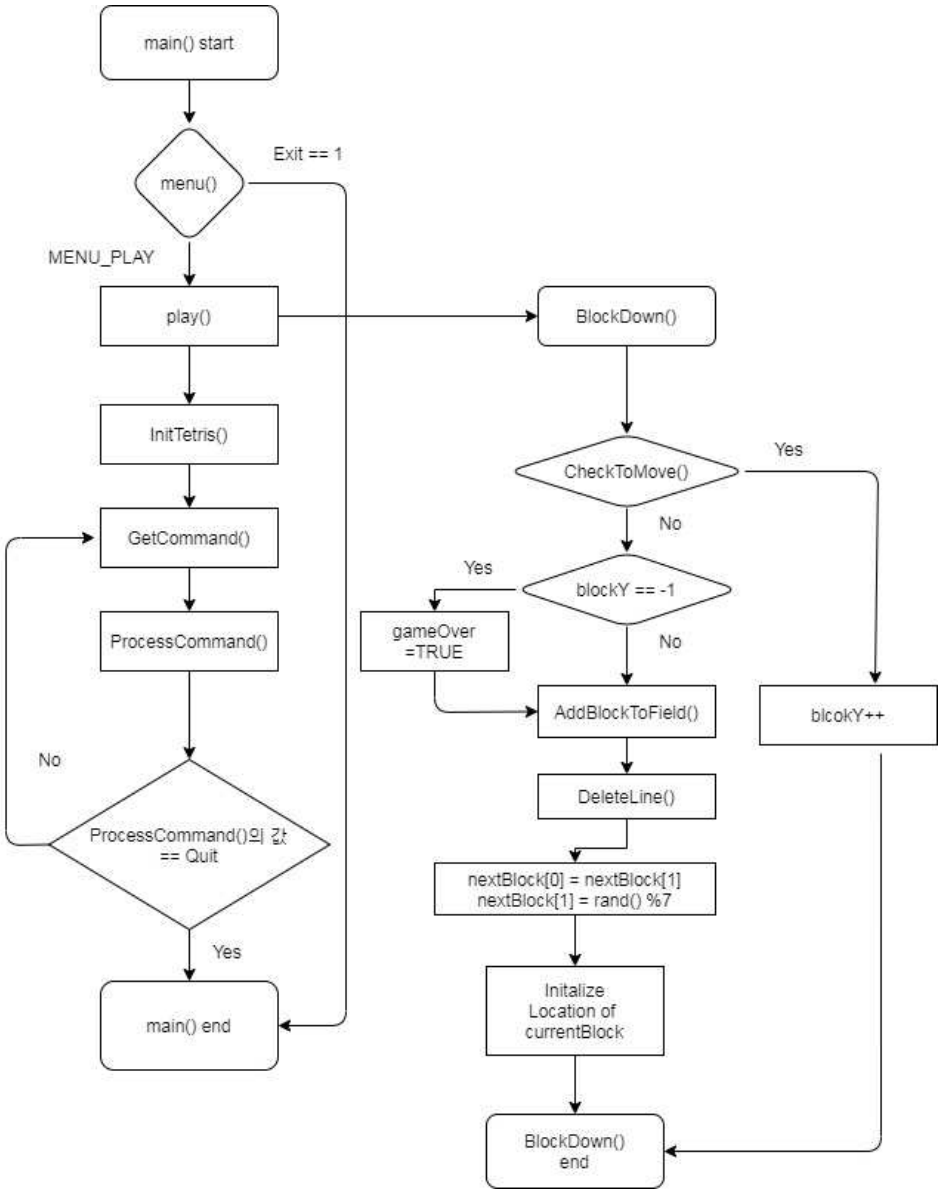

그림 3 1주차 테트리스 프로그램의 순서도

1주차에서는 테트리스 게임의 기본적인 기능을 구현한다. 프로그램을 시작하면, menu() 함수에서 메뉴를 선택하게 되는데, 1주차에서는 play 만 가능하다. MENU_PLAY를 선택하면, play() 함수가 호출되고, BlockDown() 함수는 1초에 한번씩 작동한다. 우선 BlockDown() 함수에서는 CheckToMove() 함수를 통해 블록이 아래로 이동할 수 있는지 판단하고, 만약 이동 가능하다면 블록을 한칸 아래로 내리고

종료된다. 하지만 이동 할 수 없을 때는 Y좌표가 -1이면 gameOver 변수를 True로 설정하고, -1이 아니면 AddBlockToField() 함수를 통해 블록을 필드에 추가하고, DeleteLine() 함수를 통해 삭제된 라인의 점수를 계산한다. 이후 nextBlock을 초기화하고 BlockDown() 함수가 종료된다. play() 함수에서는 InitTetris() 함수에서 필드 정보 및 블록 정보를 초기화하고, GetCommand() 함수를 통해 사용자로부터 명령을 받는다. 이후 ProcessCommand() 함수를 통해 적절한 명령을 수행한다. 만약 q/Q 키를 누르거나, gameOver 변수가 TRUE이면 게임이 종료된다.

여러 가지 함수들이 모두 제 역할을 해야만 전체적으로 게임이 흘러가므로, 각각의 호출되는 함수들의 시간복잡도를 살펴보면, CheckToMove() 함수는 블록의 크기에 따라 시간복잡도가 결정되므로 블록의 너비와 높이인 BLOCK_WIDTH, BLOCK_HEIGHT 에 따른다. 따라서 O(BLOCK_WIDTH * BLOCK_HEIGHT) 이다. 같은 이유로 AddBlockToField() 함수와 DrawChange()함수의 시간복잡도도 마찬가지로 O(BLOCK_WIDTH * BLOCK_HEIGHT) 이다. DeleteLine()함수의 시간 복잡도는 길이가 가장 긴 블록의 크기만큼의 라인이 삭제될 때 가장 시간이 많이 걸린다. 이 때 길이가 가장 긴 블록의 길이는 4이므로 4줄이 삭제될 때 가장 시간이 많이 걸린다. 또한 한 줄을 삭제할 때 필드 전체를 탐색하므로 시간 복잡도는 필드의 높이와 너비인 HEIGHT와 WIDTH로 나타낸 O(4*HEIGHT*WIDTH)이다. BlockDown()함수에서 이 함수들을 모두 호출하므로 BlockDown()함수의 시간 복잡도는 이를 모두 합친 것으로 볼 수 있다. 공간 복잡도의 경우를 살펴보면, 전체적으로 봤을 때 필드가 가장 많은 공간을 차지하므로 O(HEIGHT*WIDTH) 라고 할 수 있다.

2) 2주차

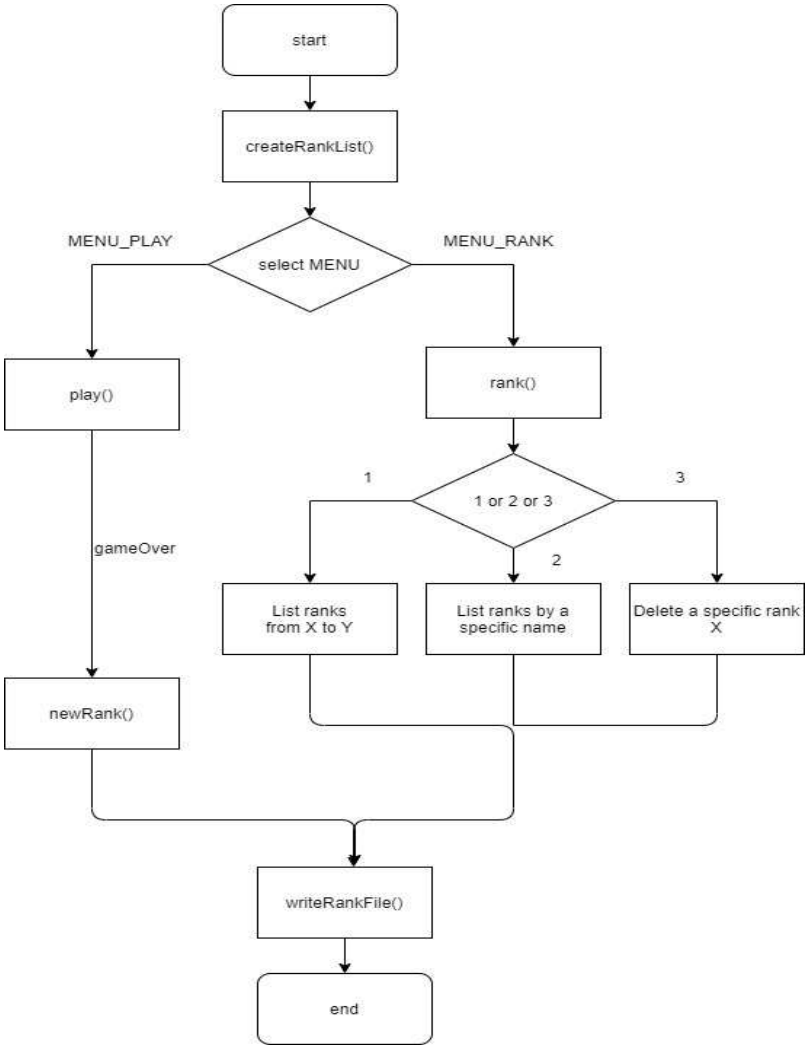

그림 4 2주차 랭킹 시스템 순서도

2주차에서는 랭킹 시스템을 구현한다. 프로그램을 시작하면 createRankList() 함수를 통해 랭킹 정보를 파일로부터 읽어들여 랭킹정보를 기록하는 자료구조에 저장한다. 이후 메뉴를 선택할 때 MENU_RANK 를 선택하면 rank() 함수가 call 되고 X부터 Y까지의 랭킹을 출력하는 기능, 특정 이름의 랭킹을 출력하는 기능, 특정 랭킹을 삭제하는 기능 중 하나를 선택할 수 있다. 혹은 MENU_PLAY 를 선택해서 게임을 플레이하면, 정상적으로 게임이 종료되었을 때 newRank() 함수가 호출되어 새로운 랭킹정보를 자료구조에 저장한다. 모든 작업이 끝나면 writeRankFile() 함수를 통해 파일에 랭킹정보를 저장해 랭킹 정보를 유지한다.

여기서 랭킹 시스템 구현을 위해 사용한 자료구조는 연결 리스트이다. 랭킹 정보를 담은 노드를 중간에 삽입하기 위해서는 연결리스트에 n개의 노드가 있다고 가정하면, 최악의 경우 모든 노드를 거쳐야 하기 때문에 O(n)의 시간복잡도가 걸린다. 또한

특정 이름이나 랭킹 정보를 토대로 랭킹 정보를 찾는 기능은 연결리스트의 탐색과 같으므로 이것 또한 O(n)의 시간복잡도를 갖는다. 특정 랭킹을 삭제하는 것도 마찬가지로 최악의 경우 O(n)의 시간복잡도를 갖는다. 공간 복잡도의 경우는 역시 n개의 노드가 필요하므로 O(n)이다.

3) 3주차

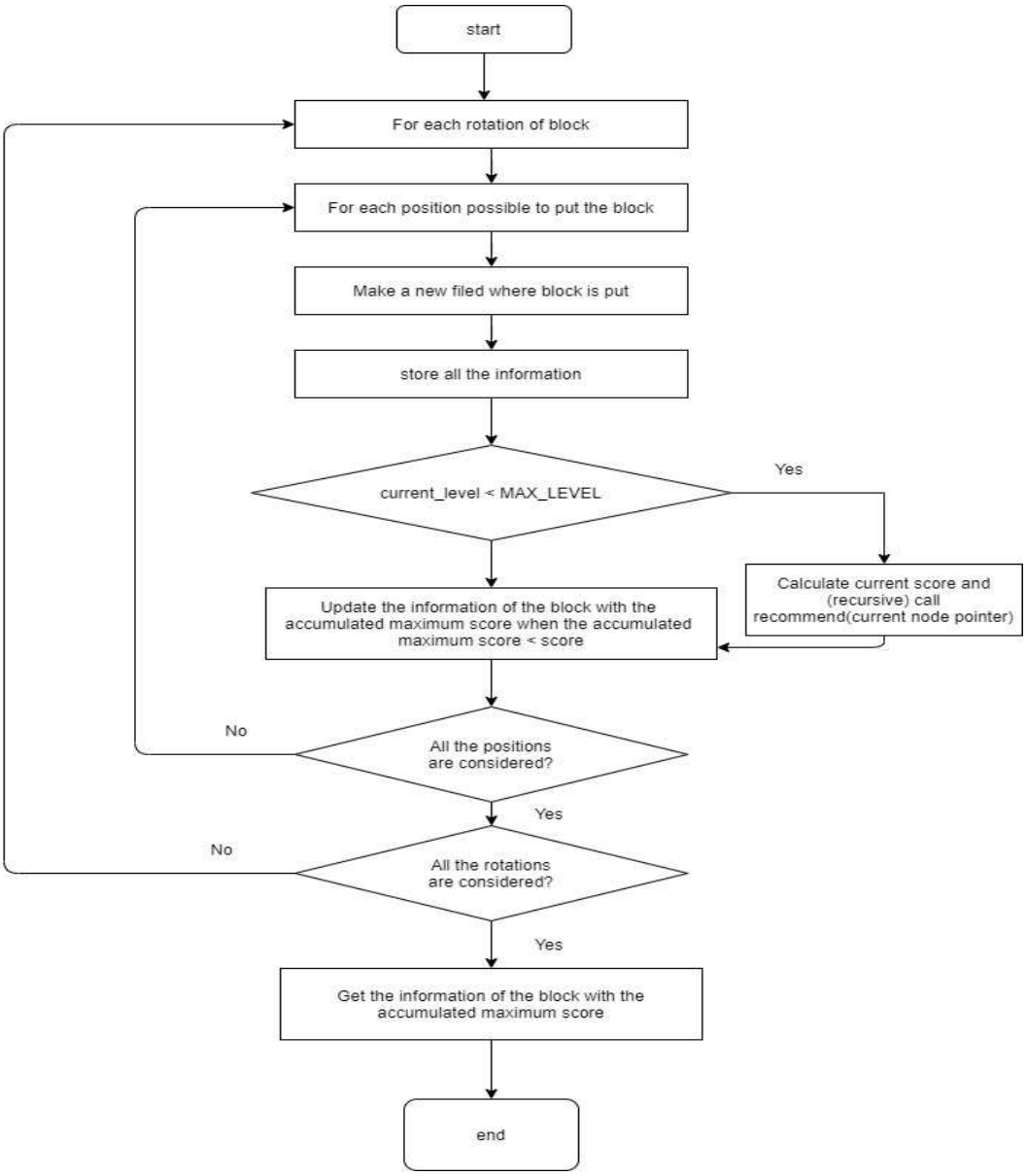

그림 5 추천기능 순서도

3주차에서는 추천 기능을 구현한다. recommend() 함수를 통해 사용자에게 좋은 점수를 얻을 수 있는 블록의 위치를 계산한다. 이후 DrawRecommended() 함수를 통해 추천 위치에 블록을 그린다. modified_recommend() 함수는 recommend() 함수보다 더 빠르고 효율적으로 추천 위치를 계산한다.
이 때 추천기능을 계산하기 위해 사용한 자료구조는 트리이다. 노드를 통해 블록의 정보들을 나타내고 노드를 트리구조로 연결시킨다. 블록이 필드에 놓일 수 있는 모든 경우의 수를 고려해야 하므로, recommend() 함수의 시간복잡도는 고려할 블록의 수 VISIBLE_BLOCKS 와 루트 노드 root의 자식노드들의 최악의 경우의 개수 34개를 통해 O(34^VISIBLE_BLOCKS)가 된다. 즉, 고려할 블록의 수가 조금만 늘어나도 시간복잡도가 매우 커지게 된다. 또한 공간 복잡도의 경우도 마찬가지로 모든 경우의 수는 그 만큼의 노드 생성을 의미한다. 따라서 공간복잡도도 O(34^VISIBLE_BLOCKS) 이다. 모든 경우의 수를 고려하는 Brute Force 알고리즘은 이 경우 성능이 매우 낮다. 따라서 pruning 알고리즘을 적용한 modified_recommend() 함수를 과제로 구현한다. modified_recommend() 함수는 자식노드들 중 가장 큰 점수를 갖는 노드만 고려하고 나머지는 가지치기 하는 pruning 알고리즘을 적용한다. 따라서 가장 큰 점수를 가진 자식노드의 자손들만 고려하게 된다. 이 때 계속해서 자식노드의 자식노드들의 비교가 이루어진뒤, 가장 큰 자식 노드의 자손들을 탐색하게 된다. 따라서 최악의 경우 자식 노드는 34개가 되고, 이 중 하나만 계속해서 그 깊이가 깊어지는 형식이므로, 시간 복잡도는 O(34*VISIBLE_BLOCKS)로 크게 개선된다. 공간 복잡도 또한 시간복잡도와 마찬가지로 O(34*VISIBLE_BLOCKS)가 된다.

2.4 제작

(1) void InitTetris()

테트리스 게임을 수행하기 위해 기본적인 변수와 자료구조를 초기화하는 함수. 테트리스 필드를 모두 채워지지 않은 상태 '0'으로 초기화하고, 현재 블록과 다음 블록의 ID를 임의로 생성한다. 현재 블록의 회전수를 0, x 좌료는 (WIDTH-2)-2, y좌표는 -1로 초기화한다. 점수를 나타내는 score와 게임이 종료되었는지 나타내는 gameover 변수, BlockDown()함수에서 사용되는 timed_out도 0으로 초기화한다. 이렇게 결정된 정보를 바탕으로 초기화면을 그린다.
input : 없음
return : 없음

(2) void DrawOutline()

블록이 떨어지는 테트리스 필드, 다음 블록이 미리 그려질 상자, 점수를 보여줄 상자의 테두리를 그리는 함수.

input: 없음
return : 없음

(3) int GetCommand()
사용자의 입력을 구분하여 입력 받은 명령을 return. KEY_UP, KEY_DOWN, KEY_LEFT, KEY_RIGHT, 'q' or 'Q' 5가지의 입력을 구분한다.
input : 없음.
return : 사용자가 입력한 명령

(4) int ProcessCommand()
GetCommand()에서 입력받은 command를 input으로 받아 적절한 동작을 취하는 함수. 적절한 동작이란 다음과 같다.

▶ QUIT : 'q' 또는 'Q'를 입력해서 게임이 강제 종료된 경우 return 값을 tetris.h 에 정의되어 있는 QUIT, 즉 'q'로 설정한다.
▶ KEY_UP : CheckToMove() 함수를 이용해서 시계 반대방향으로 90도 회전해서 화면에 그릴 수 있는지 체크한다.
▶ KEY_DOWN : CheckToMove() 함수를 이용해서 블록을 아래로 이동할 수 있는지 체크한다.
▶ KEY_LEFT : CheckToMove() 함수를 이용해서 블록을 왼쪽으로 이동할 수 있는지 체크한다.
▶ KEY_RIGHT : CheckToMove() 함수를 이용해서 블록을 오른쪽으로 이동할 수 있는지 체크한다.

그림 6 ProcessCommand()

input : int command - GetCommand() 함수에서 입력받은 명령
return : 초기에 1이지만 강제종료시엔 QUIT으로 변경

(5) void DrawField()
테트리스 필드를 그리는 함수. 필드의 정보는 0,1로 구분된다. 0은 필드가 채워지지 않음을 나타내고, 화면에 '.'로 표시됨. 1은 필드가 채워졌음을 나타내고, 공백 ' '을 attron(), attroff()를 사용해서 화면에 표시
input: 없음
return: 없음

(6) void PrintScore()
점수를 표시하는 상자안의 정해진 위치로 이동해 score 값을 화면에 출력하는 함수.

input : int score
return : 없음

(7) void DrawNextBlock()
Input parameter인 nextBlock의 정보를 바탕으로 테트리스 게임 화면에 다음 블록을 확인 할 수 있는 상자 안에 4X4인 다음 블록을 그리는 함수.
input : int *nextBlock - 블록의 ID를 저장하고 있는 배열
return : 없음

(8) void DrawBlock()
블록의 좌표, 모양, 회전수에 대한 정보들을 이용해 정해진 위치에 블록을 그리는 함수. 이때 블록은 입력 받은 모양으로 채워짐.
input : int x, int y, int blockID, int blockRotate, char tile
return: 없음

(9) void DrawBox()
상자의 왼쪽 상단의 위치, 높이와 너비를 입력받아 입력 받은 위치에 입력받은 크기를 갖는 상자를 그리는 함수.
input : int y, int x, int height, int width
return : 없음

(10) void play()
테트리스 게임을 플레이하기 위한 기본 정보를 초기화하고, 테트리스 게임을 플레이하기 위 한 전반적인 과정을 제어하는 함수.
input : 없음
return : 없음

(11) char menu()
main()함수에서 동작하는 함수로 화면에 1~4번까지 메뉴를 출력하고, 사용자가 메뉴를 선택하길 기다린다. 사용자로부터 입력을 받으면, 그 값을 return 한다.
input: 없음
return : 입력받은 메뉴 번호(character)

(12) int CheckToMove(char field[HEGHIT][WIDTH], int currentBlock, int blockRotate, int blockY, int blockX)
블록이 이동할 수 있는지 Check하는 함수. 이중 루프를 돌며 해당 위치로의 이동이 가능한지 check한다. 만약 field의 범위를 벗어나거나(0보다 작거나 WIDTH, HEIGHT 보다 클때) 이미 field가 차있을때는 0을 리턴하고, 그렇지 않다면 1을 리턴한다.

input : 필드 char field[HEGHIT][WIDTH], 현재 블록을 나타내는 int currentBlock, 블록의 회전상태를 나타내는 int blockRotate, 블록의 Y,X좌표 int blockY, blockX
return : 0 또는 1

(13) void DrawChange(char field[HEGHIT][WIDTH], int command, int currentBlock, int blockRotate, int blockY, int blockX)
블록의 위치를 바꾸는 함수. 입력으로 받은 command로 이전 블록의 상태를 알 수 있다. 만약 command가 KEY_UP이라면 회전을 한 번 한 것이므로 Rotate상태를 1 줄이고(0이라면 3으로 변경), KEY_DOWN, KEY_LEFT, KEY_RIGHT 는 반대 방향으로 좌표 정보를 수정해주면 된다. 그 후 이 정보를 이용해 이전 블록을 지우고, 현재 블록을 화면에 그려준다.
input : 필드 char field[HEGHIT][WIDTH], 명령을 나타내는 int command, 현재 블록을 나타내는 int currentBlock, 블록의 회전상태를 나타내는 int blockRotate, 블록의 Y,X좌표 int blockY, blockX
return : 없음

(14) void AddBlockToField(char field[HEGHIT][WIDTH], int currentBlock, int blockRotate, int blockY, int blockX)
블록을 필드에 추가해주는 함수. 이중 루프를 돌며 블록의 위치정보, 회전 정보, 모양 정보를 이용해 필드정보를 업데이트 해준다.
input : 필드 char field[HEGHIT][WIDTH], 현재 블록을 나타내는 int currentBlock, 블록의 회전상태를 나타내는 int blockRotate, 블록의 Y,X좌표 int blockY, blockX
return : 없음

(15) int DeleteLine(char field[HEIGHT][WIDTH])
한 줄이 꽉 찬 라인을 지우는 함수. 이중 루프를 돌며 모든 라인을 check한다. 만약 line이 꽉차지 않으면 점수 score에 포함하지 않고, line이 꽉찼을 경우에만 score를 계산해준다. 이때 score를 계산해줄 때 int cnt 변수를 이용해 지워진 line의 수를 저장하고 cnt*cnt*100을 return 해준다. 이 수가 score로서 반환된다.
input : 필드 char field[HEIGHT][WIDTH]
return : cnt*cnt*100(점수)

(16) BlockDown(int sig)
일정한 주기마다 블록을 아래로 이동하는 함수. CheckToMove() 를 통해 이동이 가능한지 판단하고 만약 이동이 가능하면 블록을 아래로 이동시키고, 그렇지 않다면 blockY가 -1 인지 판단한다. 만약 blockY가 -1이라면 더 이상 게임을 진행할 수 없으므로 gameOver 변수를 1로 설정해준다. 그렇지 않다면 블록을 필드에 추가하고 score를 계산, 현재 블록을 다음 블록으로 변경, 다음 블록을 랜덤 생성, 회전 정보와

블록의 X,Y좌표를 초기화, 다음 블록 그리기, score 출력, 필드 그리기의 명령을 순차적으로 진행한다. 모든 명령을 수행하고 전역 변수인 int timed_out 을 0으로 초기화 시켜줘 일정 주기마다 BlockDown 함수가 작동할 수 있도록 해준다.
input : int sig
return : 없음

(17) void DrawShadow(int y, int x, int blockID, int blockRotate)
블록의 그림자를 그리는 함수. 해당 블록이 필드에서 아래로 끝까지 내려갈수 있는 부분에 그림자를 그린다. 이 함수를 구현하기 위해서는 블록이 어디까지 내려갈 수 있는지 CheckToMover()함수를 이용해 y좌표를 구해서 그것을 인자로 DrawBlock()을 call하면 해결된다. 이때 char tile 인자를 '/'로 넣어주면된다.
input : 블록의 y,x좌표 int y,x, 블록의 ID int blockID, 블록의 회전정보 int blockRotate
return : 없음

(18) void DrawBlockWithFeatures(int y, int x, int blockID, int blockRotate)
블록을 그림자와 함께 그리는 함수. 간단하게 DrawShadow() 함수와 DrawBlock()을 부르고 종료된다. tetris.h 에 프로토타입을 선언해준다.
input : 블록의 y,x좌표 int y,x, 블록의 ID int blockID, 블록의 회전정보 int blockRotate
return : 없음

(19) void createRankList()
랭크 연결리스트를 만들기 위한 함수. rank.txt 파일을 읽어(rank.txt 파일이 없으면 생성) 해당 파일의 랭크 정보를 연결리스트로 구현한다. 첫 줄의 랭크의 수를 전역변수 N에 저장하고, Node에 해당 랭크에 해당하는 이름과 score를 저장하고, 연결되게 만들어 연결리스트를 구축한다.
input : 없음
return : 없음

(20) void rank()
특정 이름을 입력받아 해당 이름의 랭크 정보를 출력하는 기능과 특정 랭크번호를 입력해 그 랭크를 삭제하는 기능이 추가된 rank() 함수. 사용자로부터 특정 이름을 입력받으면 연결리스트의 처음부터 탐색하며 해당 이름과 같은 이름을 가진 노드를 만나면 해당 노드의 정보를 출력한다. 이때 연결리스트의 탐색은 다음 노드를 가리키는 포인터 Node* next를 통해 할 수 있다. 또한 현재 노드를 가리키는 Node* cur을 통해 탐색을 진행한다. 모든 노드를 탐색하였으나 해당 이름을 찾지 못하였다면 에러

메시지를 출력한다. 사용자로부터 삭제할 랭크정보를 받으면 입력받은 랭크가 적절한지 판단한다. 만약 입력받은 정수가 랭크의 수인 N보다 크거나 0보다 작으면 오류메시지를 출력한다. 적절한 정수라면 count를 이용해 연결리스트의 처음부터 탐색을 진행한다. 이때 삭제를 위한 Node* prev를 선언해주는데, 이것은 단일 연결리스트의 삭제를 위한 cur의 이전 노드를 가리키는 포인터다. cur은 현재 방문한 노드를 가리키고, prev는 cur의 이전 노드를 가리킨다. 만약 연결리스트에서 해당 랭크를 찾았을 때 cur이 head와 같다면, 즉 첫노드가 해당 랭크라면, 그 노드의 다음 노드가 있는지 없는지 판단한다. 만약 다음 노드가 없다면 그대로 메모리를 해제해주고 head, tail을 모두 NULL 로 초기화해준다. 다음 노드가 있다면 head를 cur->next 를 가리키게 하고, cur->next는 NULL로 초기화해준뒤 메모리 해제해준다. 만약 첫 노드가 아닌 N보다 작은 곳에서 해당 랭크를 찾으면, prev->next 가 cur->next를 가리키게 해주고, cur을 메모리해제한다. 만약 마지막 노드가 해당 랭크라면, tail이 prev를 가리키게 하고, cur을 메모리해제한다. 이렇게 하면 연결리스트를 유지하면서 해당 랭크를 삭제할 수 있다. 이러한 복잡한 방식은 더블링크드리스트를 이용하면 쉽게 해결할 수 있지만, 여기서는 cur의 이전 노드를 가리키는 prev를 이용해 해당 문제를 해결한다.
input : 없음
return : 없음

(21) void writeRankFile()
랭크 정보를 파일에 기록하기 위한 함수. rank.txt 파일에 랭크 정보를 기록한다. 연결리스트의 처음부터 끝까지 탐색하며 모든 랭크 정보를 rank.txt 파일에 기록한다.
input : 없음
return : 없음

(22) void newRank(int score)
테트리스 게임이 gameOver 되었을 경우 새로운 랭크 정보를 기록하기 위한 함수. 사용자로부터 이름을 입력받아 해당 이름과 score를 연결리스트에 추가한다. 이때 연결리스트는 score를 기준으로 내림차순으로 정렬되게 Node를 추가한다. Node에 이름 및 score정보를 기록해 연결리스트에 추가한다. 모든 기록이 완료되면 writeRankFile() 함수를 call해 rank.txt에 랭크 정보를 유지한다.

(23) int recommend(Node * root)
모든 경우의 수를 조사해 블록, 블록의 회전수, 필드의 상태, 누적 score를 갖는 노드로 구성된 모든 play 시퀀스를 표현하기 위한 트리를 만드는 함수. 호출될 때 마다 root->c에 child가 연결된다. 만약 현재 고려하는 레벨(lv)가 최대 레벨(VISIBLE_BLOCK+1) 보다 작으면, 누적 score를 계산하기 위해 재귀적 함수 호출을 한다. 반면 현재 고려하는 레벨이 최대 레벨과 같다면 현재 score를 저장한다. 누적

score가 가장 큰 경로 상에 존재하는 현재 블록의 추천된 위치와 회전수를 기억하여 블록의 위치를 추천한다. 트리를 구성하기 위해 RecNode** child를 이용해 root에 재귀적으로 연결시킨다. 첫번째 루프는 RecRotate를 이용해 가능한 회전의 경우를 모두 고려한다. 두번재 루프는 가로의 경우 놓일수 있는 경우를 모두 조사한다. 즉 CheckToMove함수를 이용해 이동가능한 곳인지 판단하고 조사한다. child[idx]에 메모리를 할당하고, 레벨, 블록ID, 회전 정보 등 을 기록하고, Y좌표를 내릴 수 있는 곳까지 내린다. 이 때 갱신된 필드정보를 child[idx]에 저장하고, X,Y 좌표와 누적 스코어도 갱신한다. 만약 root->lv+1이 고려할 블록 수와 같다면 max를 갱신하고 그렇지 않다면 임시 max인 tempMax를 recommend함수를 호출해 계산한다. 만약 tempMax가 max보다 크거나 같다면, max 를 tempMax로 설정한다. 이때 tempMax와 max가 같을 경우, Y좌표가 더 작을 경우에만 max를 갱신하도록 한다. 이렇게 하지 않으면 max와 tempMax가 같을 경우 블록이 간헐적으로 오른쪽(루프의 끝부분)으로 쏠리는 현상이 발생한다. Y좌표를 낮게 잡으면 블록이 쏠리지 않고 고르게 추천된다. 다만 이 방법의 문제점은 일자형 블록이 세로로 들어가는 경우가 거의 없고, 가로로만 추천된다는 점이다. 이는 과제 때 해결할 수 있도록 한다.
input : 루트노드가 될 RecNode* root
return : 누적점수중 가장 큰 점수 int max

(24) void DrawRecommend(int y, int x, int blockID, int blockRotate)
추천 블록을 필드에 그려주는 함수. X,Y좌표와 블록 ID, 블록 회전정보가 파라미터로 주어지므로 기존에 있던 DrawBlock 함수에 char 'R' 을 파라미터에 추가해서 불러주기만 하면 된다.
input : int y, int x, int blockID, int blockRotate
return : 없음

(25) int modified_recommend(RecNode* root)
테트리스 블록을 높은 점수를 얻을 수 있는 위치에 추천해주는 함수. 기존 Recommend()함수와는 다르게 모든 경우의 수를 고려하지 않는다. int prunMax 라는 변수를 추가해 두 번째 루프에서 가장 큰 score를 prunMax로 설정한다. 이 prunMax보다 작은 score를 갖는 노드의 자식들은 고려하지 않게한다. 즉, 그 노드는 가지를 치는 Pruning 알고리즘을 적용한다. 그 외의 부분은 recommend()함수와 동일하다. 기존 recommend() 함수는 주석처리한다.
input : 루트노드 RecNode* root
return : int max

(26) void recommendPlay()
자동으로 추천 블록으로 블록을 이동시키며 사용자의 입력 없이도 테트리스 게임을

플레이하게 해주는 함수. play() 함수와 거의 동일하지만 act.sa_handler 가 새롭게 추가한 recBlockDown 함수 인 점이 다르다.
input : 없음
return : 없음

(27) void recBlockDown(int sig)
자동으로 추천 블록으로 블록을 이동시키는 함수. BlockDown()함수와 비슷하다. 다만 modified_recommend()함수로부터 얻은 정보를 이용해 바로 추천 위치로 블록을 이동시킨다.
input: 없음
return : 없음

2.5 시험

(1) 1주차

작성한 함수들이 작동을 제대로 하는 지 확인하기 위해 3가지 Case를 확인하였다.

Case 1	command 입력이 없는 경우
Case 2	한 줄이 완성 되었을 경우
Case 3	gameOver의 경우

1) Case 1

사용자가 아무런 입력을 하지 않아도 블록은 일정 주기마다 아래로 이동한다. 또한 그 상태로 계속 방치했을 때, 블록이 필드의 범위를 넘어서면 다음과 같이 gameover 문구가 뜨며 종료된다.

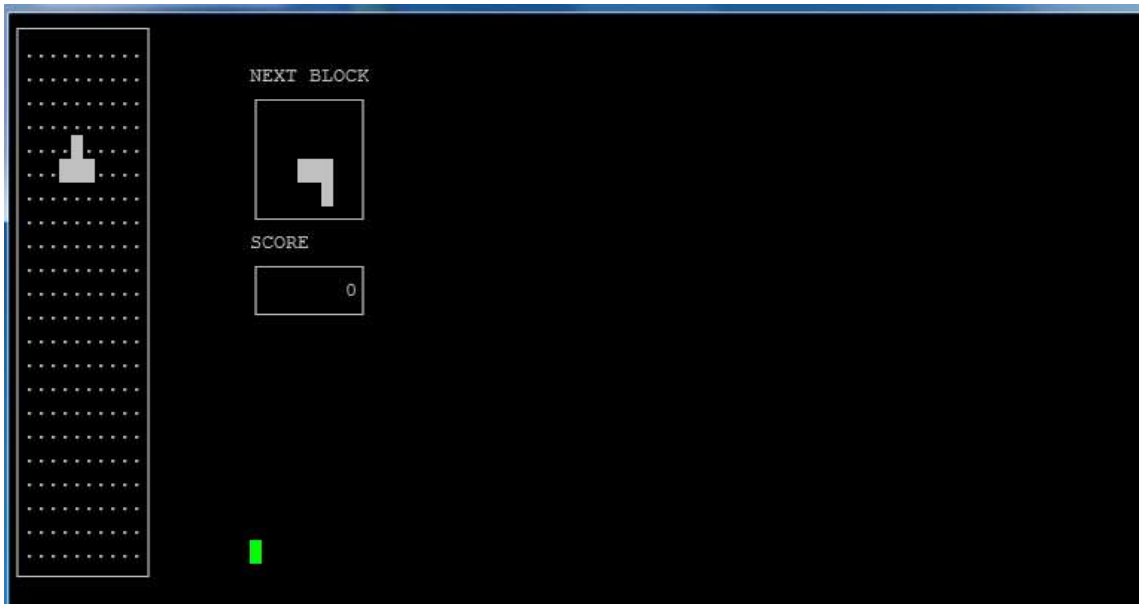

그림 7 Case1 command 입력이 없는 경우(1)

그림 8 command 입력이 없는 경우(2)

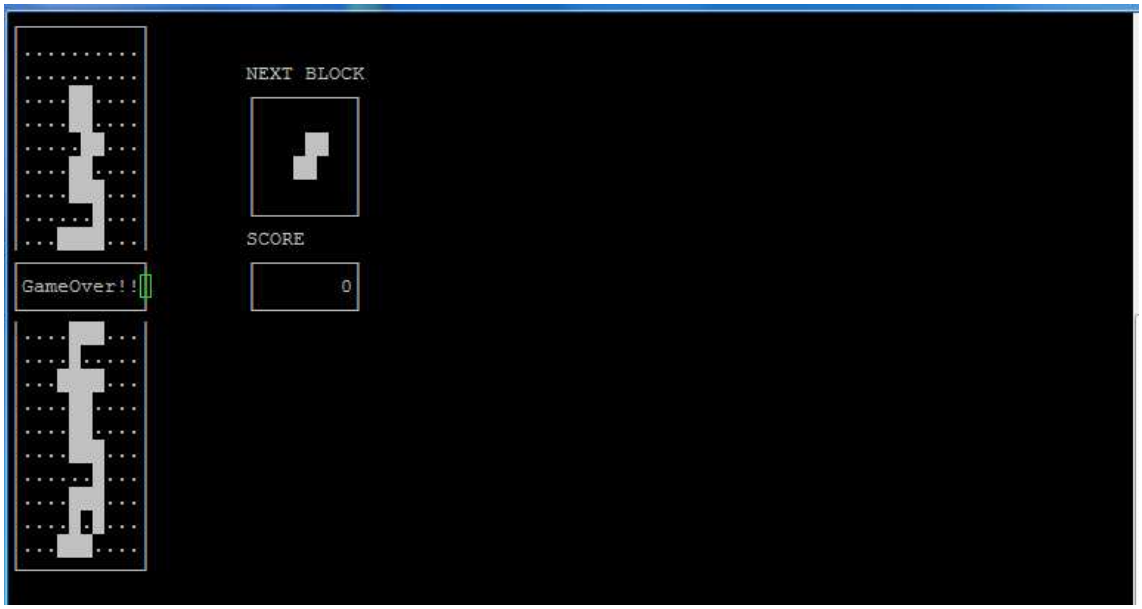

그림 9 command 입력이 없는 경우(3)

[그림 9]과 같이 gameover가 되는 것을 확인 할 수 있다. 다만 기존 시중에 있는 테트리스 게임은 게임오버를 일으키는 마지막 블록도 필드에 표시되나, 위 프로그램은 표시가 되지 않고 gameOver된다. 그러나 tetris_sample 프로그램 또한 똑같은 결과가 나오므로 위 프로그램은 조건을 만족한다고 볼 수 있다.

2) Case 2

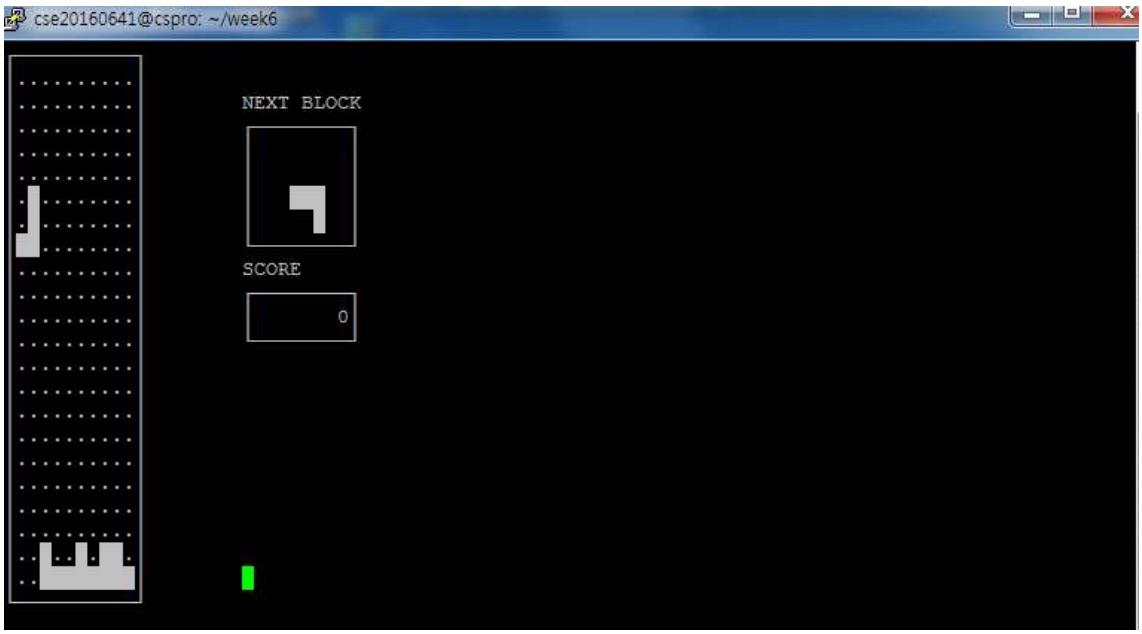

그림 10 한 줄이 완성되었을 경우(1)

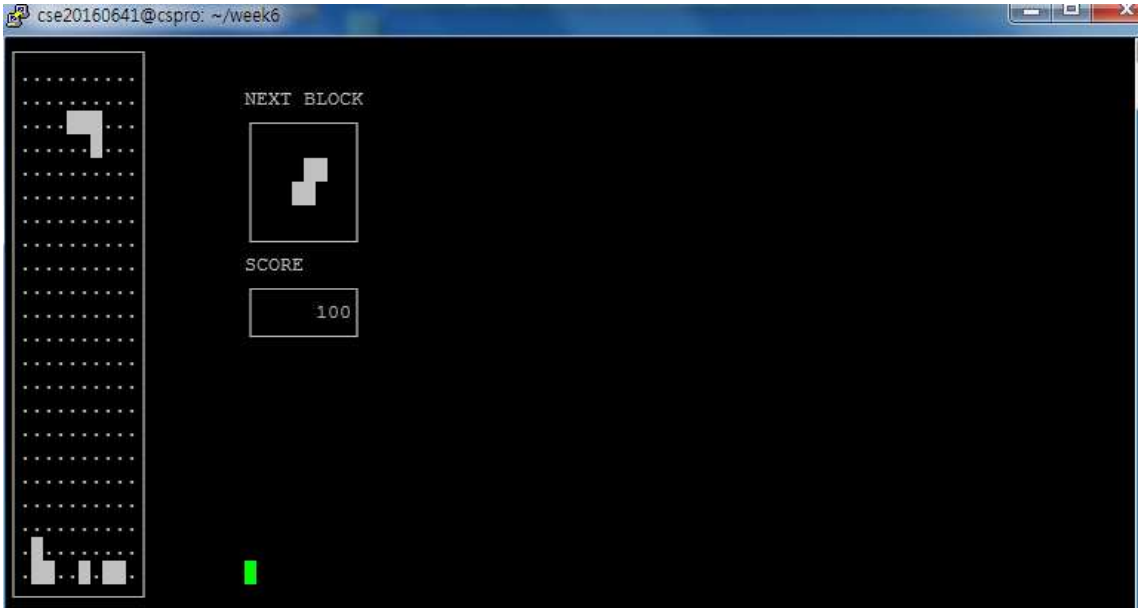

그림 11 한 줄이 완성되었을 경우(2)

[그림 10]에서 현재 블록이 그 위치에서 아래로 쭉 내려와 한 줄이 완성 되었다. 그 후 완성된 한 줄이 삭제되고 위에 있던 줄이 아래로 내려오고, score가 100 증가하는 것을 확인 할 수 있다.

3) Case 3

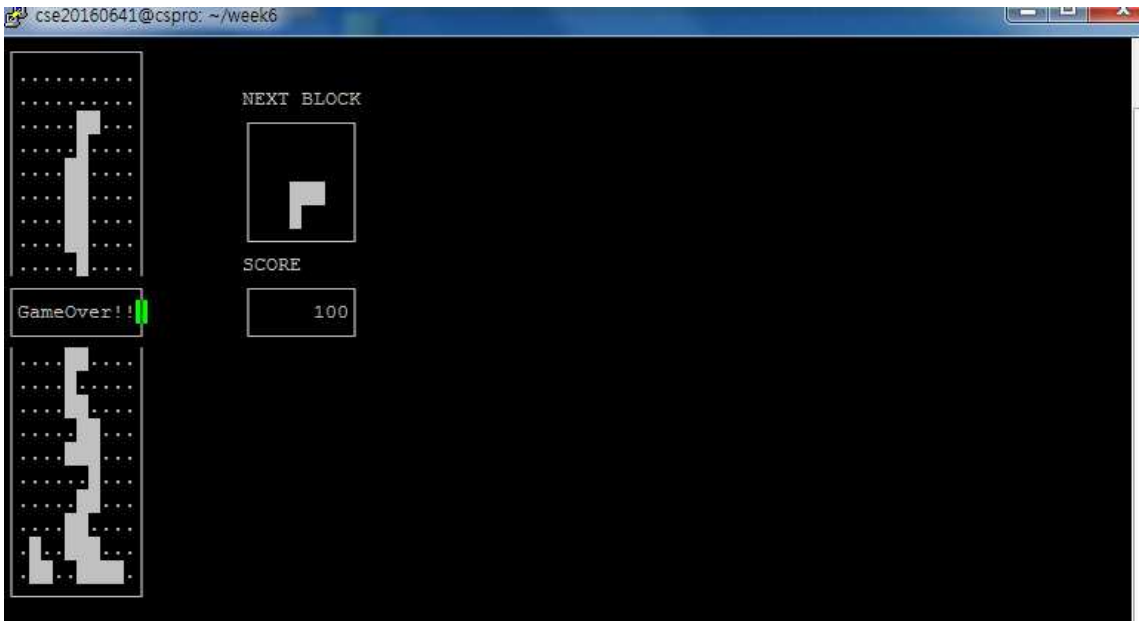

그림 12 gameOver의 경우

위와 같이 블록이 더이상 쌓일 수 없는 경우에는 gameover 문구가 뜨며 게임이 종료되는 것을 확인 할 수 있다.

위와 같이 3가지 경우를 살펴본 결과, 작성한 5개의 함수들이 모두 제 기능을 다하는 것을 확인 할 수 있다. 또한 다음을 통해 그림자 기능, 2개의 블록 미리 보여주기, 닿은 면적만큼 score 증가 가 제대로 구현되었는지 확인 할 수 있다.

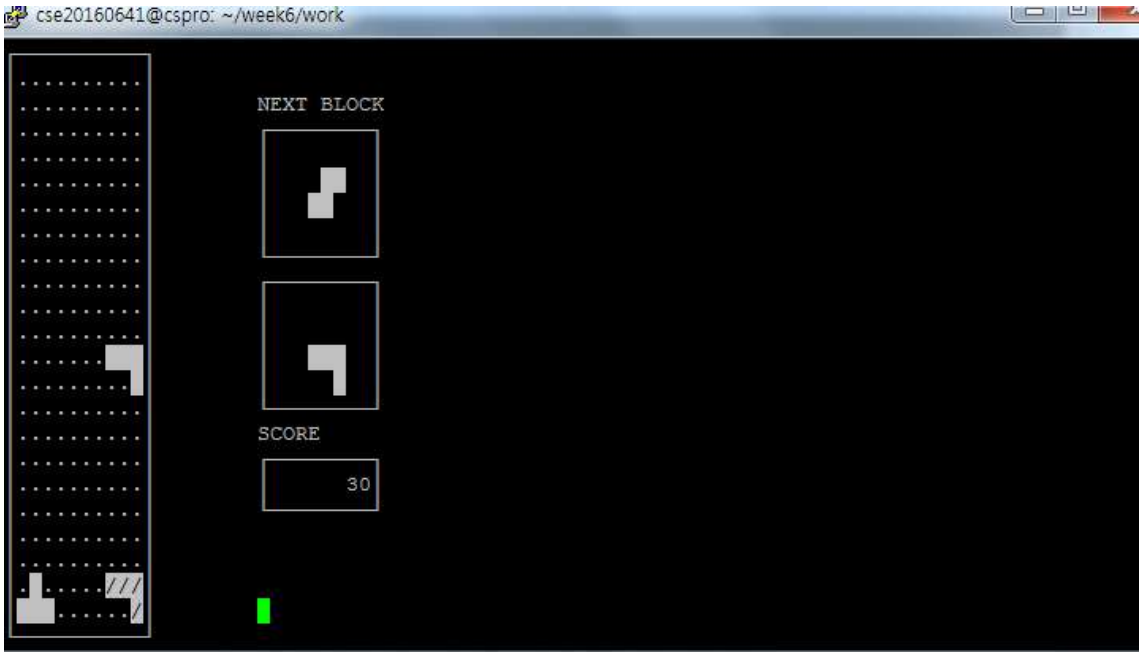

그림 13 결과(1)

위와 같이 2개의 블록이 미리 출력되는 것을 확인 할 수 있다. 또한 블록이 아래로 떨어질것을 그림자로 통해 확인 할 수 있다. 또한 필드와 블록이 맞닿아서 점수가 30 증가한 것을 확인 할 수 있다.

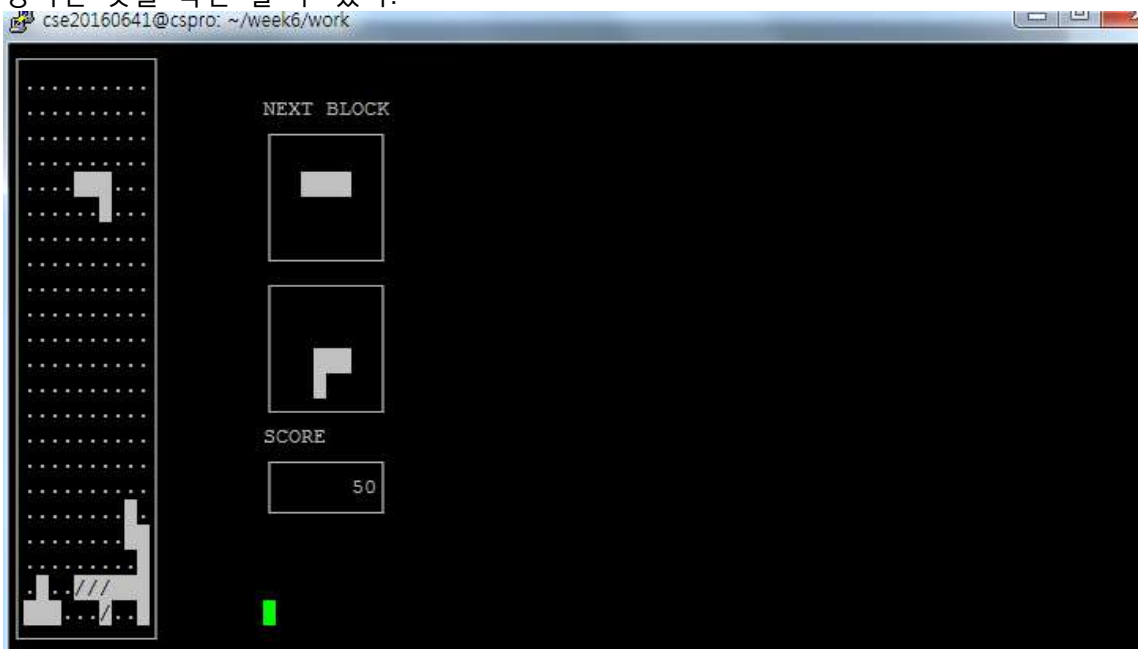

그림 14 결과(3)

여기서 블록이 필드에 닿은 4곳과, 블록위에 블록이 맞닿은 곳 1곳으로 인해 점수는 50이 된것을 확인 할 수 있다. 블록이 필드에 닿은것만 점수에 포함하는 것으로 이해했으나, sample_tetris 에서 블록과 블록이 맞닿은(y좌표로) 부분도 점수에 포함하기에 같은 방식을 적용했다.

위와 같은 결과들을 통해 그림자 기능, 2개의 블록 미리 보여주기, 닿은 면적만큼 score 증가하기 등 3가지의 기능이 정상적으로 작동하는 것을 확인 할 수 있다.

(2) 2주차

그림 15 랭크 기능 선택 화면

메뉴 선택에서 2를 입력하면 랭크 기능으로 넘어가진다. 이 화면에서는 아직 2, 3번 구현은 되지 않아 2, 3,을 입력할 경우 다시 메뉴선택창으로 넘어간다. 1번을 선택할 경우에는 아래와 같은 화면이 나온다.

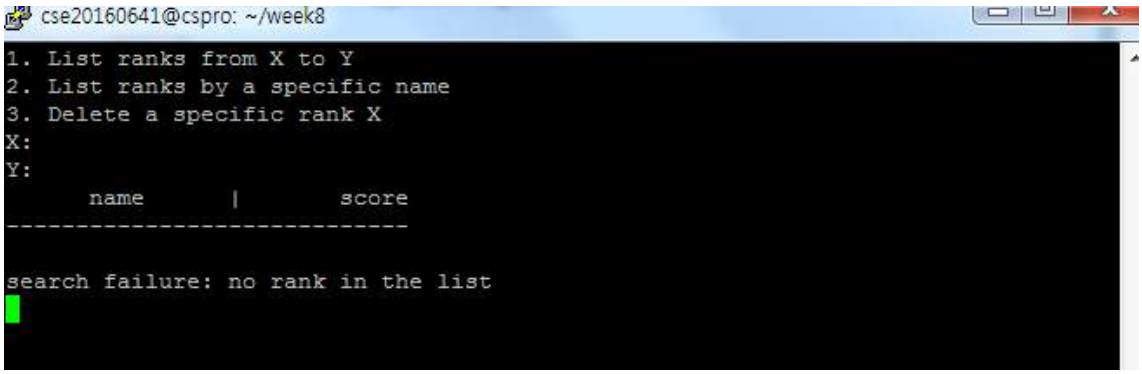

그림 16 X와 Y 입력창

X와 Y를 입력받고, 위와 같은 결과가 나오는데, 이는 현재 rank.txt 조차 생성되어 있지 않기 때문에 오류메시지가 발생하는 것이다. 따라서 우선 게임을 진행해 랭크 정보를 생성하도록 한다.

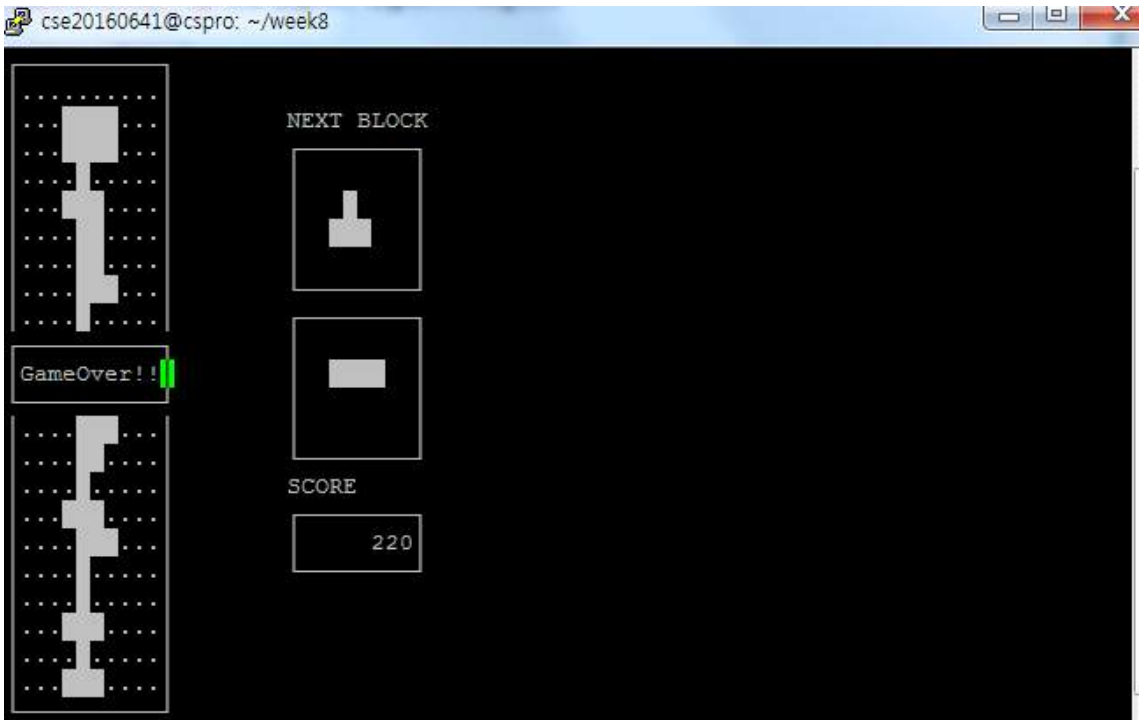

그림 17 gameOver 화면

그림 18 score 입력 화면

위와 같이 gameOver가 될 경우 사용자로부터 score를 입력받을 수 있다. 이렇게 score를 입력하고 다시 rank 기능으로 돌아가면 아래와 같은 결과가 나온다.

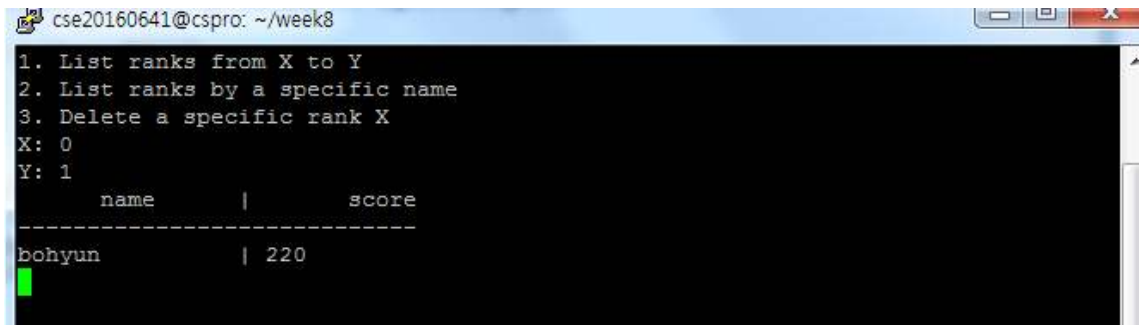

그림 19 rank 기능 구현

위와 같이 새로운 rank 정보가 입력되었음을 확인 할 수 있다. 또한 다시 한 번 게임을 플레이하고 더 높은 점수를 받으면 다음과 같은 결과 나온다.

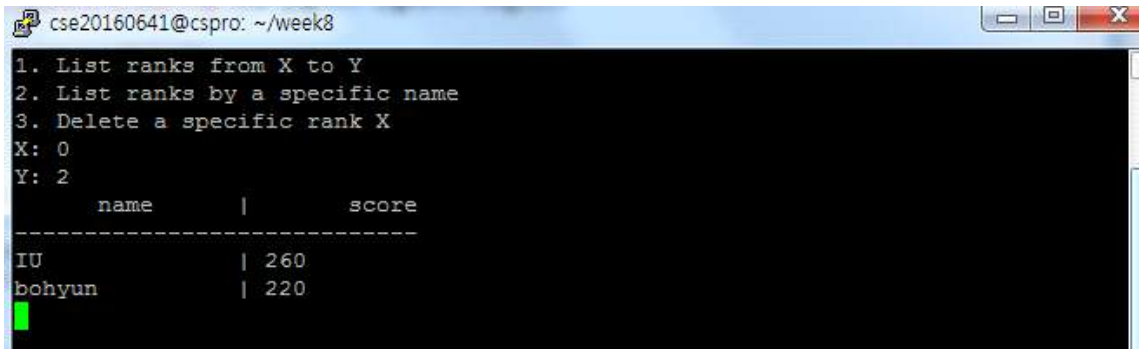

그림 20 score 정렬 확인

위와 같이 score가 내림차순으로 정렬되는 것을 확인 할 수 있다. 이 때 범위를 설정해서 랭크를 확인 할 수 있는데, 그 결과는 다음과 같다.

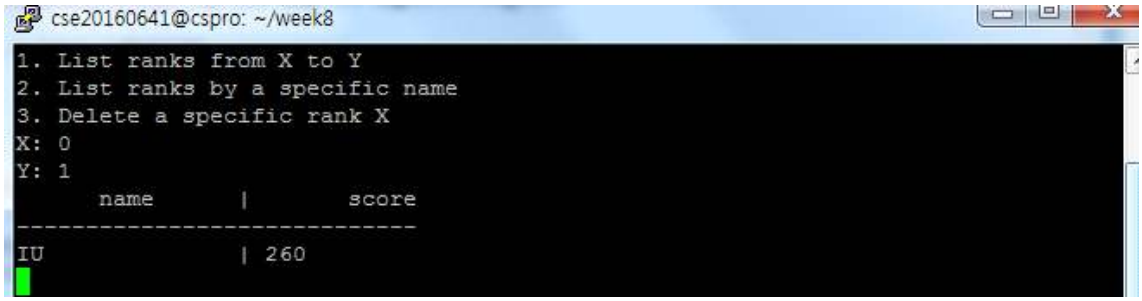

그림 21 범위에 따른 결과 출력

0부터 1까지의 결과만을 출력하는 것을 확인 할 수 있다.

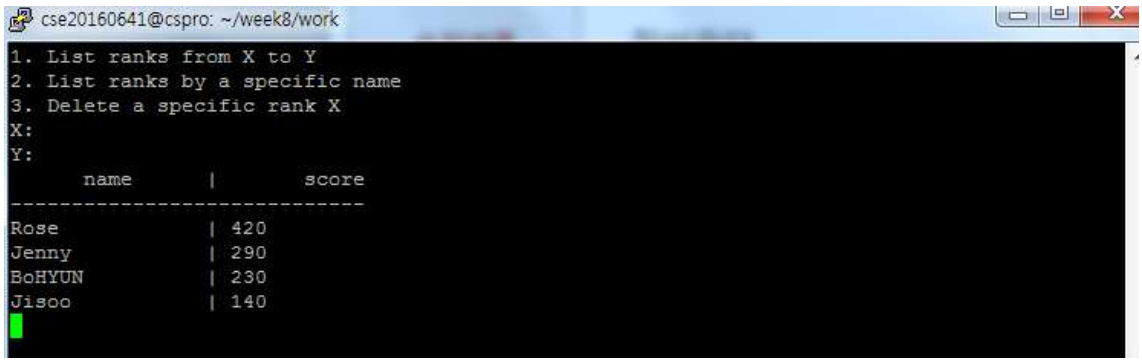

그림 22 현재 랭크 정보

결과를 분석하기 위해 4개의 랭크정보를 생성한 뒤, 각 기능을 테스트 해본다. 특정 이름의 랭크 정보를 탐색하는 기능의 결과는 다음과 같다.

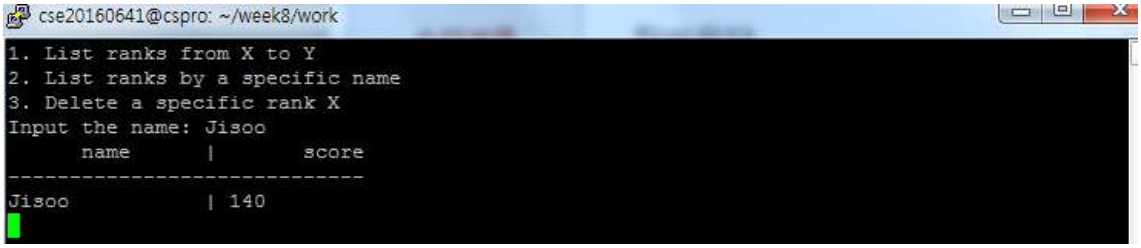

그림 23 특정 이름을 통해 랭크 정보 찾기

위와 같이 Jisoo 의 랭크 정보를 찾아 적절히 출력하는 것을 확인 할 수 있다.

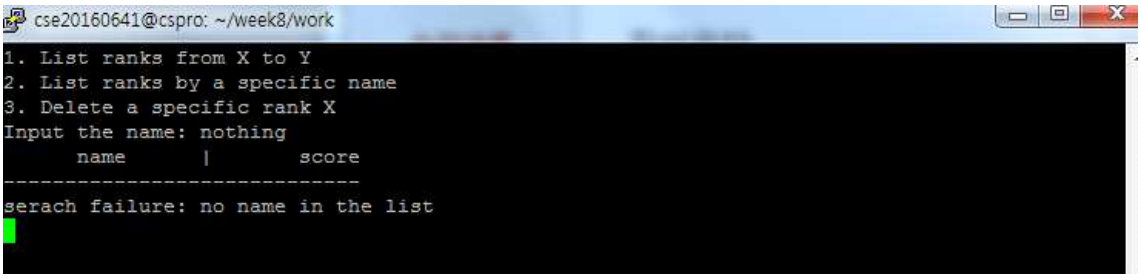

그림 24 존재하지 않는 특정 이름 정보 찾기

위와 같이 존재하지 않는 이름을 검색하면 오류메시지를 출력한다.

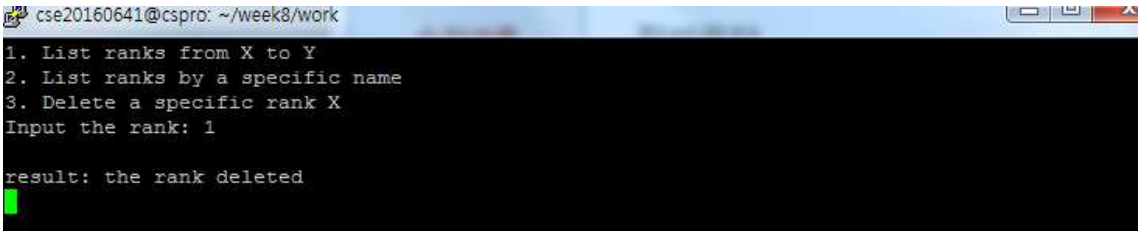

그림 25 첫번째 노드 랭크 삭제(1)

그림 26 첫번째 노드 랭크 식제(2)

첫번째 랭크 정보를 삭제하면, 위와 같이 첫번째 랭크인 Rose가 삭제되고 3개의 랭크만 남는 것을 확인 할 수 있다.

그림 27 중간 노드 랭크 삭제(1)

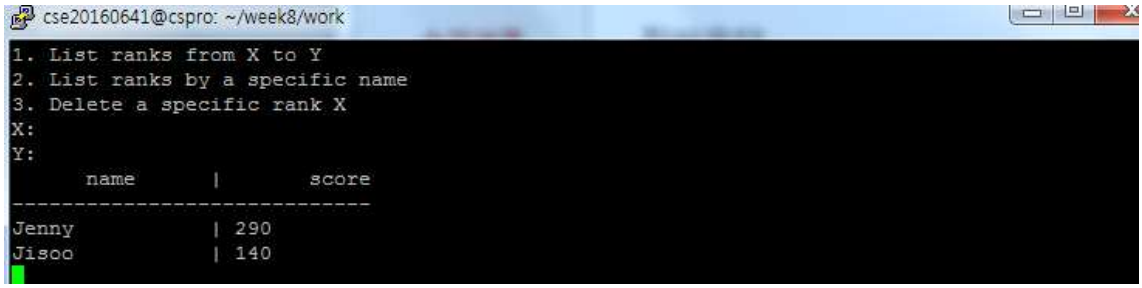

그림 28 중간 노드 랭크 삭제(2)

중간에 위치한 BoHYUN 랭크가 삭제되고 2개의 랭크만 남은 것을 확인 할 수 있다.

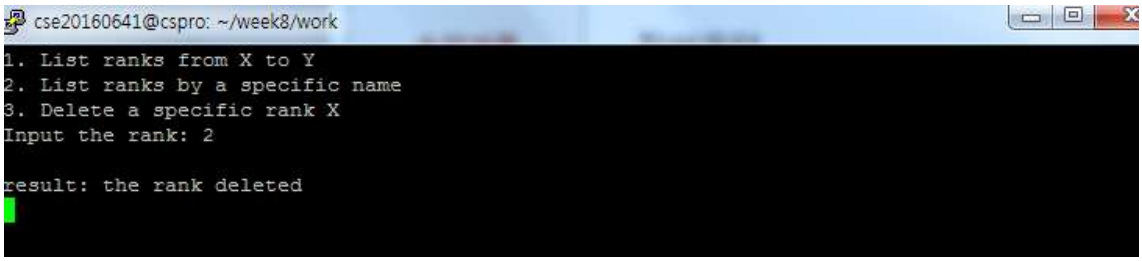

그림 29 마지막 노드 랭크 삭제(1)

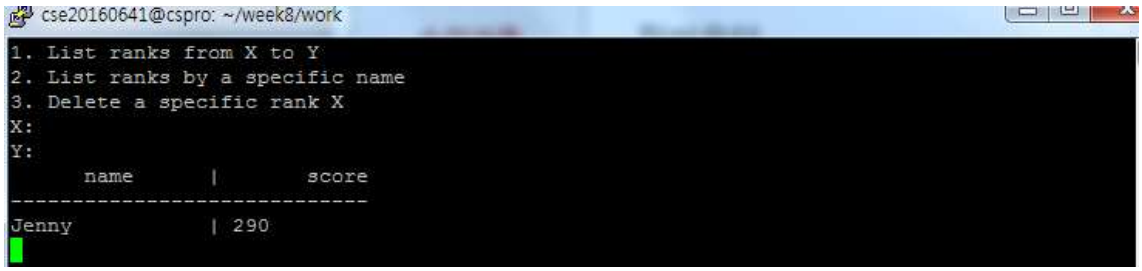

그림 30 마지막 노드 랭크 삭제(2)

마지막 노드의 랭크가 삭제된 것을 확인 할 수 있다. 이렇게 모든 위치의 랭크가 적절히 삭제되는 것을 확인 할 수 있다.

그림 31 랭크 삭제 오류

위와 같이 존재하지 않는 랭크를 삭제하려 하면 에러 메시지가 출력된다. 위 결과들을 통해 프로그램이 적절히 구현되었음을 확인할 수 있다.

(3) 3주차

고려할 블록의 수가 많아질수록 그 정확도는 상승하고 얻을 수 있는 점수가 커지게 될 것이다. 하지만 고려할 블록의 수가 많아질수록 시간 복잡도 및 공간복잡도가 커지므로 프로그램이 매우 느려지고 성능이 떨어지게 될 것이다. 이를 테스트해보기 위해 고려할 블록의 수를 세가지 경우로 나누었다.

Case 1	VISIBLE_BLOCK 2
Case 2	VISIBLE_BLOCK 3
Case 3	VISIBLE_BLOCK 4

1) Case 1

첫 번째로 VISIBLE_BLOCK을 2으로 설정하고 테트리스를 실행한다. 이는 고려할 블록의 수가 두 개라는 뜻이다.

그림 32 Case 1 VISIBLE_BLOCK 2

고려할 블록의 수를 2로 설정하고 테트리스를 실행하면, 약 4분정도의 플레이타임을 갖고, 점수는 6100이 나왔다. 중간중간 엉뚱한 곳에 블록을 추천하는 경우가 잦았으며, 추천기능이 강력하다고 보기 힘들다.

2) Case 2

그림 33 Case 2 VISIBLE_BLOCK 3

고려할 블록의 수를 3으로 설정하고 테트리스를 실행하면, 약 15분정도 플레이타임을 갖고, 점수는 13970이 나오게 된다. 중간중간 엉뚱한 곳에 블록을 추천하기도 하지만 Case1에 비해 그 빈도가 낮다. 따라서 꽤 준수한 점수를 얻을 수 있는 것을 확인 할 수 있다.

3) Case 3

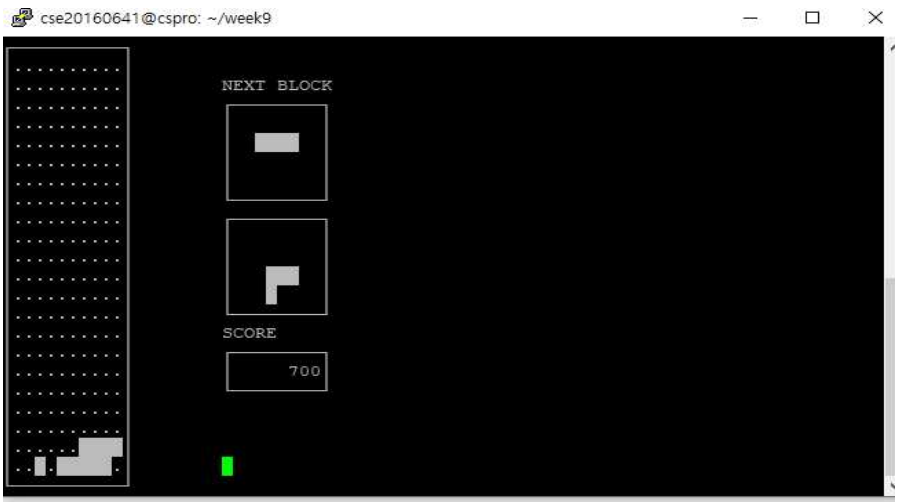

그림 34 Case 3 VISIBLE_BLOCK 4

고려할 블록의 수를 4로 설정하고 테트리스를 플레이하면, 그 속도가 매우 느려진다. 블록하나가 나오는데 약 8초정도가 걸려, 점수를 몇 점까지 얻을 수 있을지 테스트해 보기가 어려울 수준이 된다.

이러한 비효율성을 개선하기 위해 pruning 알고리즘을 적용한 추천기능 modified_recommend의 결과는 다음과 같다. 이 결과 또한 모든 경우의 수를 고려했을 때와 같이 VISIBLE_BLOCK 의 값을 변경시켜가며 결과를 비교한다. 또한 recommendPlay()기능을 이용해 결과를 빠르게 확인한다.

Case 1	VISIBLE_BLOCK 2
Case 2	VISIBLE_BLOCK 3
Case 3	VISIBLE_BLOCK 4

1) Case 1

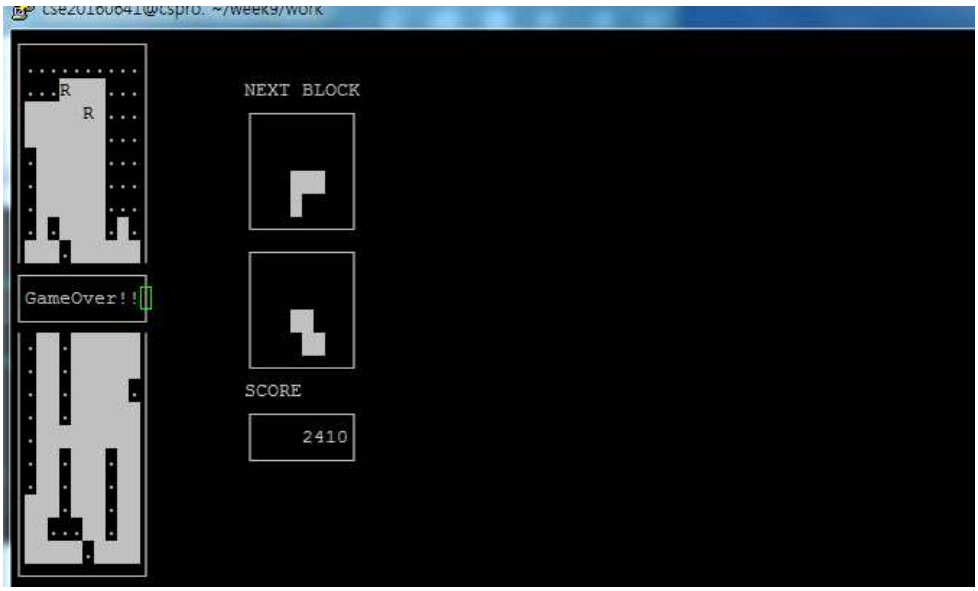

그림 35 Case1 VISIBLE_BLOCK 2

고려할 블록의 수가 두 개일 경우, 그 성능은 매우 낮다. 2410점이라는 낮은 점수를 기록하고 게임 오버된다. 플레이 타임은 약 22초로, 게임오버되기까지의 시간은 매우 짧았다.

2) Case 2

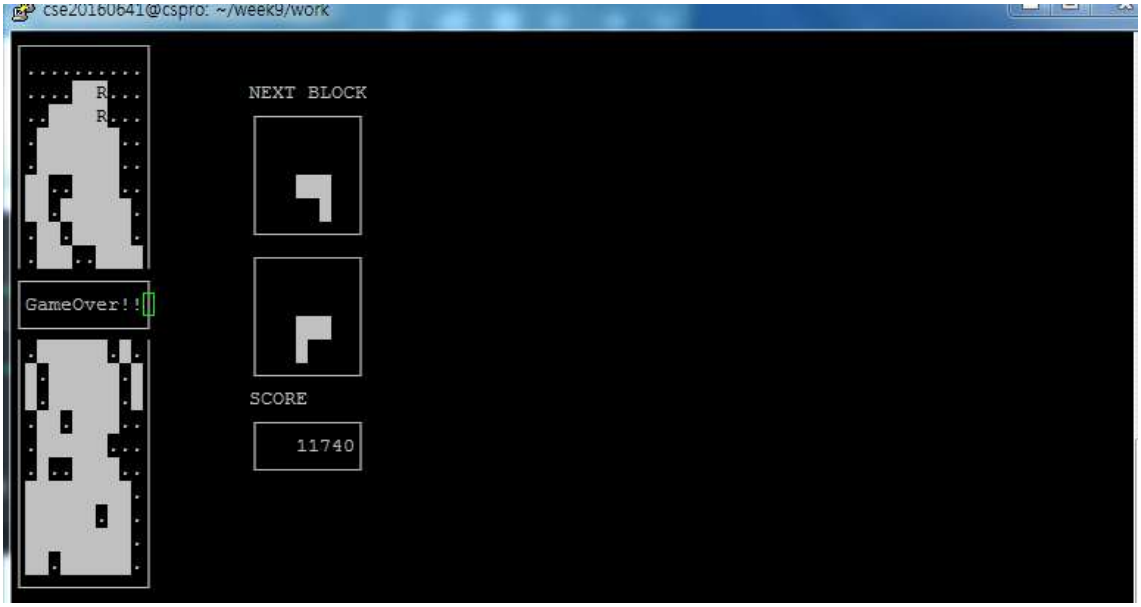

그림 36 Case2 VISIBLE_BLOCK 3

고려할 블록의 수가 세 개로 늘어나면, 그 성능은 훨씬 나아진다. 모든 경우의 수를 고려했을 때의 결과 13970과 비교하면 여전히 점수는 낮지만, 그 속도는 훨씬 빠른 것을 확인 할 수 있다.

3) Case 3

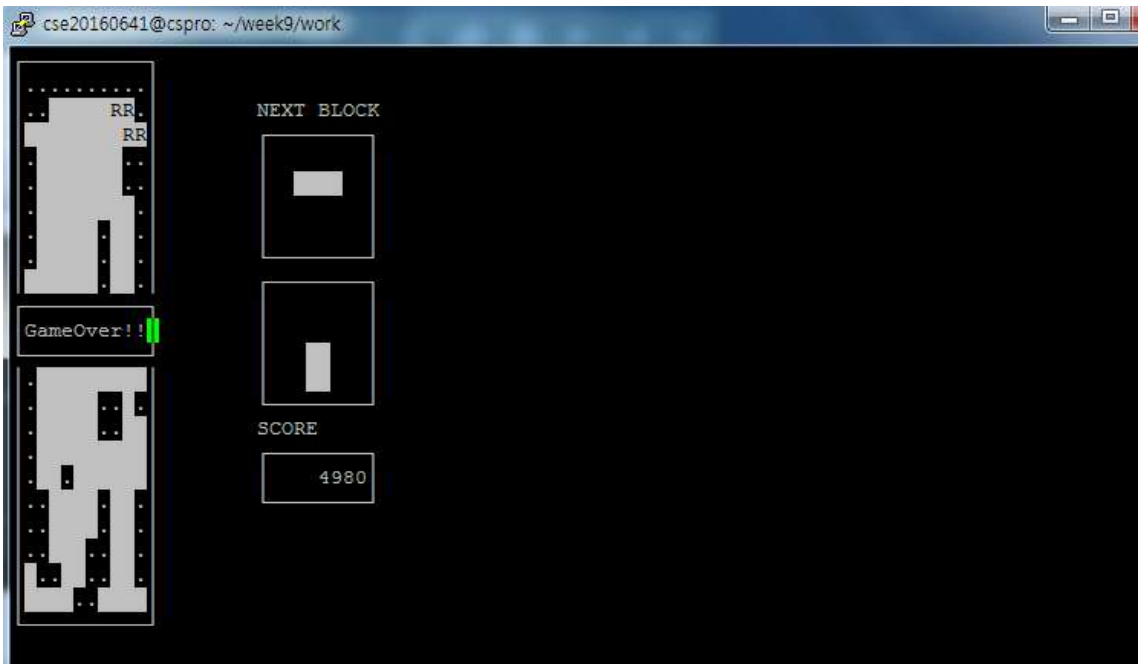

그림 37 Case3 VISIBLE_BLOCK 4

고려할 블록의 수를 4개로 늘리면, 특이하게도 그 성능이 더 낮아진다. 점수는 4980점을 기록했고, 몇 번을 더 돌려봐도 비슷한 결과가 나옴을 확인 할 수 있다. 이에 대한 정확한 이유는 파악하기 힘드나, 이는 Pruning 알고리즘을 적용한 위 프로그램의 큰 문제점이라고 할 수 있다. 다만, 걸리는 시간은 매우 짧아진다. 모든 경우의 수를 고려할 때는 고려할 블록의 수가 네 개를 넘어가면 결과를 체크하기조차 힘들정도로 시간이 오래 걸렸으나, Pruning 알고리즘을 적용한 경우에는 3분의 시간도 걸리지 않았다.

2.6 평가

순서도를 기반을 프로그램을 구현하였기에 순서도와 실제 구현 사이의 차이가 거의 없다. 또한 전체적으로 실습과 과제에서 제시한 지시사항들을 충실히 이행하고 조건들을 만족하는 결과들을 보여주어 실험의 목표는 충분히 달성했다고 볼 수 있다. 다만 3주차의 modified_recommend() 함수에서 더 정확하고 효율적인 알고리즘을 구현하였으면 더 좋았을 것이라는 점은 아쉬운 점이다.

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

테트리스 프로그램은 주차별로 추가되거나 수정되는 부분이 있다. 이러한 변경 사항을 정확히 반영하여 프로그램의 모양이 일관성 있게 유지되도록 주차별 폴더를 나누어 파일을 관리하고, Tetris라는 통합 폴더에서 소스 파일들을 Git 프로그램을 이용해 버전을 관리했다.

2.9 보건 및 안정

1) 1주차

1주차에서는 getCommand() 함수에서 사용자로부터 명령을 입력받아 processCommand() 함수에서 특정 명령을 수행하게 된다. 이때 특정 명령에는 q/Q - 프로그램 종료, KEY_DOWN - 블록 아래로 한칸 이동, KEY_UP - 블록 회전, KEY-LEFT, KEY_RIFHT - 왼쪽 혹은 오른쪽으로 블록 이동 등이 있다. 이러한 특정 명령을 제외한 다른 명령을 입력하면 프로그램이 오류가 발생할 수 있다. 이를 처리

하기 위해 default 문, 즉 특정 명령을 제외한 모든 명령은 무시하는 문장을 삽입한다. 따라서 이 dafault 문으로 인하여 오류는 발생하지 않게 된다.

2) 2주차

2주차에서는 오류가 발생 할 수 있는 상황이 매우 많다. 먼저 처음 프로그램을 실행하면 createRankList() 함수에서 파일로부터 랭킹정보를 읽어들여야 하는데, 이때 파일이 존재하지 않으면 에러가 발생할 수 있다. 이는 인풋 파일인 "rank.txt"가 존재하지 않으면 생성하도록 처리하면 오류가 발생하지 않는다. X부터 Y까지의 랭킹정보를 출력하는 기능에서도 X와 Y의 범위가 적절하지 않으면 오류가 발생할 수 있다. X가 0보다 작거나, Y가 0보다 작거나, X가 Y보다 크거나, X가 Y보다 큰 경우에 오류가 발생한다. 이를 방지하기 위해 범위를 적절하게 입력하지 않으면 오류 메시지를 출력하도록 설정해준다. 또한 X의 초기값은 1로, Y의 초기값은 N(랭킹정보 노드의 개수)으로 설정해두면, X혹은 Y값을 사용자가 입력하지 않아도 오류가 발생하지 않고 랭킹정보를 출력할 수 있다.

3) 3주차

3주차에서는 실습에서 만든 프로그램의 경우, 모든 경우의 수를 고려하기 때문에 많은 수의 노드가 필요해 공간복잡도가 매우 크다. 이 때 너무 많은 노드가 생성되면 메모리가 부족하여 오류가 발생할 수 있다. 또한 시간복잡도도 매우 크기 때문에 프로그램이 끝나지 않는 오류가 발생할 수 있다. 이를 방지하기 위해서는 일정 수 이상의 메모리가 소요되면 강제로 프로그램을 종료하거나, 일정 시간 이상이 지나면 프로그램을 종료하도록 해야 한다. 과제에서 만든 프로그램도 마찬가지로 고려해야할 블록의 수가 늘어날수록 메모리소요와 시간소요가 커지기 때문에 똑같은 방식을 적용해주어야 한다.

3. 기 타

3.1 환경 구성

1) gcc

gcc는 유닉스 환경에서 많이 쓰이는 c언어 컴파일러이다.[1] 이번 테트리스 프로젝트에서는 이 gcc를 이용해 소스파일을 컴파일한다. gcc의 다양한 옵션은 다음과 같다.

2) gcc 옵션

(1) -o name : 실행 파일명을 name으로 대치한다. 이 옵션이 없는 경우 gcc는 기본적으로 실행 파일 명을 a.out으로 한다.

(2) -S : 컴파일 과정에서 어셈블 이전 과정만 수행한다. 실행 결과 해당 C소스파일의 어셈블코드인 .S 파일이 생성된다.

(3) -c : 컴파일 과정에서 linking 이전 과정만 수행한다. 실행 결과 해당 C소스파일의 목적파일인 .o 파일이 생성된다.

(4) -Idir : include 파일을 찾을 대 해당 dir에서 검색한다. include 파일이 현재 디렉토리에 존재하지 않는 경우 이 옵션을 필요로 한다.
(5) Dmacro(=defn) : gcc에서 macro를 선언한다.
(6) -w : 컴파일시 발생하는 모든 경고메시지를 출력하지 않는다.
(7) -wall : 경고 메시지뿐만 아니라 의문시되는 코딩 습관에 대해서도 경고를 출력한다.
(8) -g : 디버깅을 위한 정보를 포함하여 컴파일 한다. gdb를 사용하기 위해서는 이 옵션을 꼭 사용하여 컴파일 해야 한다.

3) gdb

gdb는 GNU 디버거, 즉 GNU 소프트웨어 시스템을 위한 기본 디버거이다. C,C++, 포트란 등의 여러 프로그래밍 언어를 지원하는 다양한 유닉스 기반의 시스템에서 작동하는 이식성 있는 디버깅 툴이다.[2]

4) vi 에디터

vi 에디터는 유닉스 환경에서 가장 많이 쓰이는 문서 편집기다. vi 에디터는 명령모드와 편집모드가 있으며 처음에는 일반적으로 명령모드로 시작하게 된다. 명령모드에서 I키를 누르면 편집모드를 사용할 수 있으며, 편집모드에서 문서내용을 추가 하거나 삭제하는 등 편집 작업을 할 수 있다. 이 때 ESC 키를 누르면 편집모드를 빠져나올 수 있다. 이 상태에서 ':' 키를 누르고 wq를 입력하면 저장하고 종료를 할 수 있다.[3]

5) Ncurses 라이브러리

Ncurses 라이브러리는 텍스트 모드에서 Window, Panel, Menu, Color, Mouse 등을 쉽게 사용할 수 있도록 도와주는 라이브러리이다. gcc를 설치하면 자동으로 같이 설치된다. 이를 사용 하기 위해서는 헤더파일을 소스파일에 포함해주면 된다. 컴파일 시에는 -lncurses 라는 옵션을 추가해주어야 한다. ncurses 라이브러리의 내용은 다음과 같다.

(1) WINDOW *initscr()

ncurses library를 사용 하기 위해 가장 먼저 호출되는 함수. ncurses 의 자료구조를 초기화한다.
input : 없음
return : WINDOW

(2) int endwin()

endwin()는 ncurses library를 사용한 프로그램에서 intscr()함수를 호출하기 전 상태로 모드를 바꾸고 커서를 좌측 하단으로 이동시킨다. endwin()함수를 호출하기전 newwin()함수에 의해 열려진 모든 윈도우를 delwin()함수를 통해 닫아야 한다.
input : 없음
return: int

(3) int noecho(void)
사용자가 입력한 타이핑이 화면에 출력되지 않도록 하는 함수.
input : 없음
return : int

(4) int keypad(WINDOW* win, int bf)
bf가 TRUE 이면 윈도우 win에서 입력을 받을시 일반 stdin에서 지원하지 않는 키패드 및 특수문자의 입력을 가능케 한다. Ncurses는 키패드의 기능과 화살표 키들을 위해 ncurses.h에 정의된 키 코드를 반환한다.
input : WINDOW* win, int bf
return : int

(5) WINDOW* newwin(int nlines, int ncols, int y, int x)
(y,x) 위치에 nline, ncols의 크기를 가지는 윈도우를 생성하여 해당 윈도우의 포인터를 return. 생성에 실패할 시 NULL을 리턴.
input : int nlines, int ncols, int y, int x
return : NULL 또는 WINDOW*

(6) int delwin(WINDOW* win)
윈도우 win을 더 이상 사용하지 않을 경우 윈도우를 지워주는 함수. 프로그램 종료시 newwin()을 사용하여 생성한 모든 윈도우를 delwin()을 통해 삭제해야 함.
input : WINDOW* win
return : int

(7) int move(int y, int x)
커서를 y, x 로 이동시켜주는 함수. 이를 이용해 원하는 좌표에 출력을 할 수 있다.
input : int y, int x
return : int

(8) int wmove(WINDOW* win, int y, int x)
move() 함수와 유사하게 윈도우 win의 좌표 (y,x)로 커서를 이동하는 함수.
input : WINDOW* win , int y, int x
return : int

(9) char getch(void)
키보드 입력을 받아 입력받은 charater를 return 하는 함수.
input : 없음.

return : char

(10) char wgetch(WINDOW* win)
윈도우 win에서 키보드 입력을 받아 입력받은 character를 return 하는 함수

(11) int addch(chtype ch)
문자 ch를 화면의 현재 커서의 위치에 출력하는 함수.
input : ncurses에서 정의된 character chtype ch
return : int

(12) printw(WINDOW* win, const char* format[, argument , ...])
format 매개변수로 서식화 된 일련의 argument를 ncurses에서 출력시 사용하는 screen인 stdscr상에서 출력해주는 함수.
input : WINDOW* win, const char* format[, argument , ...]

(13) int waddch(WINDOW* win, chtype ch)
문자 ch를 win 화면의 현재 커서의 위치에 출력하는 함수.

(14) int wprintw(WINDOW* win, const char* format[, argument ,..])
format 매개변수로 서식화 된 일련의 argument를 윈도우 win에 출력하는 함수.
input : WINDOW* win, const char* format[, argument , ...]

(15) int attoff(int attr)/int attron(int attr)
atton, attoff 사이에 출력되는 문자의 색깔을 attr 모드로 출력하는 함수. 아래와 같은 모드를 가지고 있다.

mode	description
A_NORMAL	Normal display (no highlight)
A_STANDOUT	Best highlighting mode of the terminal.
A_UNDERLINE	Underlining
A_REVERSE	Reverse video
A_BLINK	Blinking
A_DIM	Half bright
A_BOLD	Extra bright or bold
A_PROTECT	Protected mode
A_INVIS	Invisible or blank mode
A_ALTCHARSET	Alternate character set
A_CHARTEXT	Bit-mask to extract a character
COLOR_PAIR(n)	Color-pair number n

그림 38 attr 모드

input : int attr

return : int

(16) int box(WINDOW* win, int vert, int hor)
win의 외곽에 테두리를 그려주는 함수.
input : WINDOW* win, int vert, int hor
return : int

3.2 참고 사항

[1] 서강대학교 컴퓨터공학과 교수진, "컴퓨터공학 설계 및 실험 I", 서강대학교 공학부 컴퓨터공학과, 2012

[2] 위키피디아, “GNU디버거” https://ko.wikipedia.org/wiki/GNU_%EB%94%94%EB%B2%84%EA%B1%B0)

[3] 위키피디아, “vi” , https://ko.wikipedia.org/wiki/Vi

3.3 팀 구성

조보현 100%

3.4 수행기간

2020년 10월 13일 ~ 2020년 11월 9일