2020년도 2학기 컴퓨터공학설계및실험I

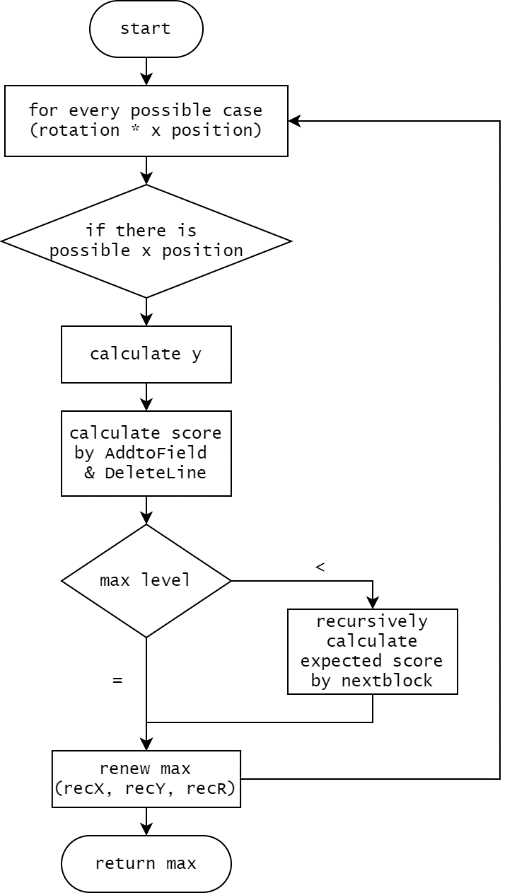
9주차 결과 보고서

20170175 김태안

1. 실습 목적

테트리스 게임을 플레이하는 사용자를 위해 어떤 위치에 블록을 놓으면 높은 점수를 받을 수 있을지를 추천하는 추천 시스템을 구현한다. 추천된 블록은 1주차 숙제의 그림자 기능을 이용해서 화면상에 나타낸다. 이 과정을 통해서 tree와 포인터(pointer)에 대한 이해를 높일 수 있다.

1. 실습 구현 내용



추천 시스템은 위와 같은 알고리즘을 통해 추천 위치와 예상 점수를 계산한다. 이는 recommend 함수를 통해 구현되었다.

int recommend (RecNode \*root);

입력에 대해 가장 많은 점수를 얻을 수 있는 위치를 찾는 함수이다. 블록과 필드의 정보가 담긴 RecNode를 입력받은 뒤, 가능한 모든 다음 블록의 회전과 위치 경우의 수에 대해 점수를 계산한다. 이때, 현재 탐색 깊이가 최대 깊이가 아니라면, 그 다음 블록이 놓일 상황도 재귀적으로 계산해 가장 큰 점수를 얻을 수 있는 위치를 구한다. 계산한 값이 여태까지 계산한 최댓값보다 크다면 추천 위치를 갱신해 모든 위치에 대한 탐색 후 추천 위치를 출력한다. 탐색이 끝나면 현재 노드에 연결된 자식 노드를 모두 해제한다.

void DrawRecommend (int y, int x, int blockID, int blockRotate);

추천 위치에 블록을 R 문자를 사용해 출력한다. 이 함수는 DrawBlocksWithFeatures에서 호출되어 추천 블록을 필드에 그린다.

void BlockDown(int sig);

추천 시스템을 사용할 수 있도록 BlockDown 함수를 수정한다. nextBlock은 MAX\_LEVEL에 맞는 크기를 가진다. 예를 들어, 깊이가 4인 탐색을 한다면, 총 5개의 블록을 미리 알고 있어야한다. 한 블록이 쌓이면 blockDown에서 nextBlock의 맨 앞의 원소를 제거하고 원소의 순서를 조정한다. 그리고 새 블록의 모양을 결정한다. nextBlock을 조정한 뒤, recommend를 호출해 추천 위치를 계산하고 출력한다.

위 추천 시스템은 현재 상태에 대해 Number of Rotation \* Width 만큼의 경우의 수를 계산한다. 그리고 각 경우의 수는 Number of Rotation \* Width 만큼의 경우의 수를 또 계산한다. 이는 MAX\_LEVEL 만큼 반복된다. 즉, 사용되는 노드의 수는  
이다. 따라서 공간 복잡도는 O(wrm)이다. recommend 함수는 O(wrm)번 노드를 생성하며 AddBlockToField, DeleteLine, CheckToMove를 호출한다. 이들은 BLOCK\_HEIGHT \* BLOCK\_WIDTH 또는 HEIGHT \* WIDTH의 시간 복잡도를 가진다. 이를 16, 220의 상수로 가정하면 recommend의 시간 복잡도는 O(wrm)이다.

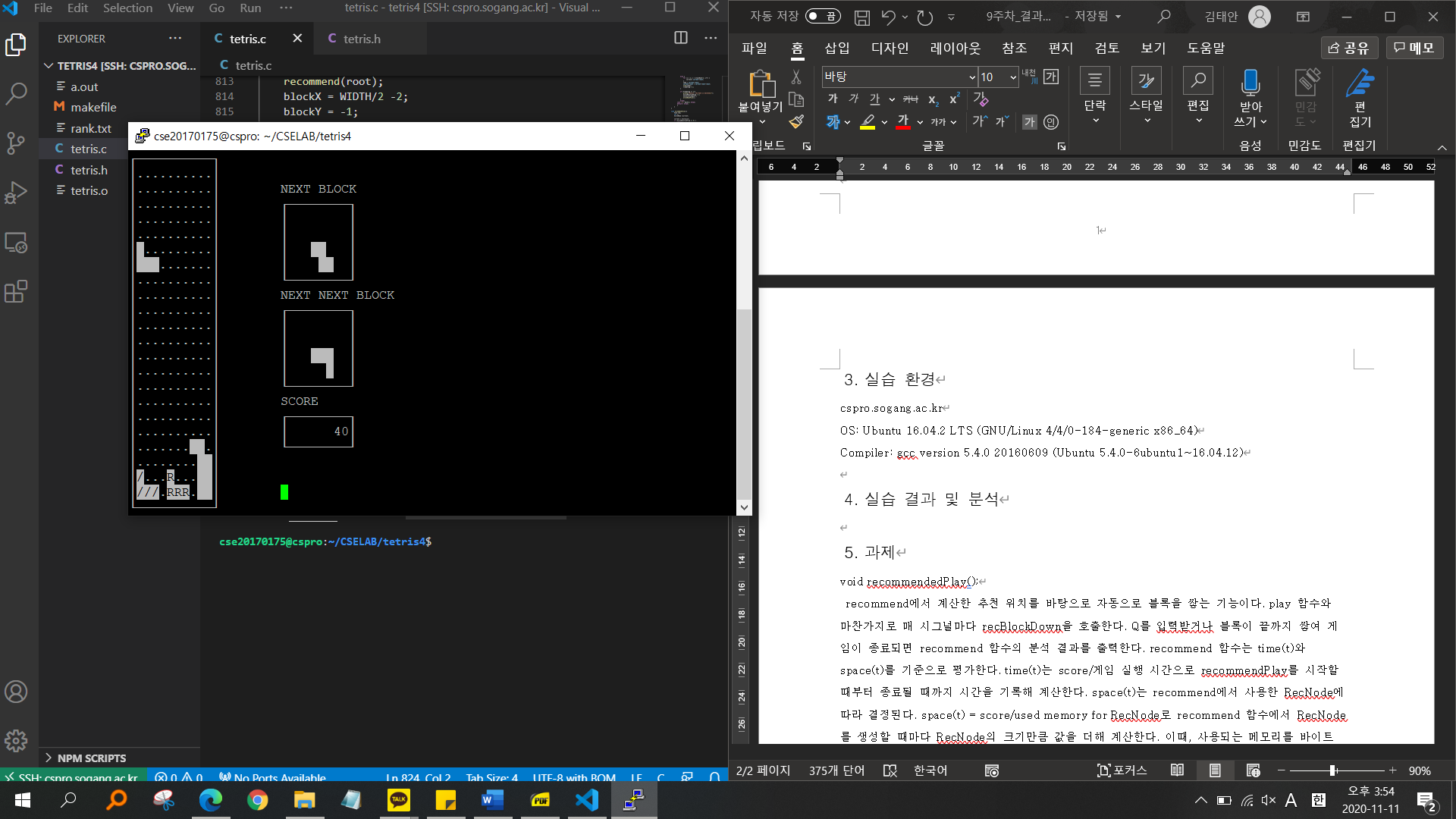
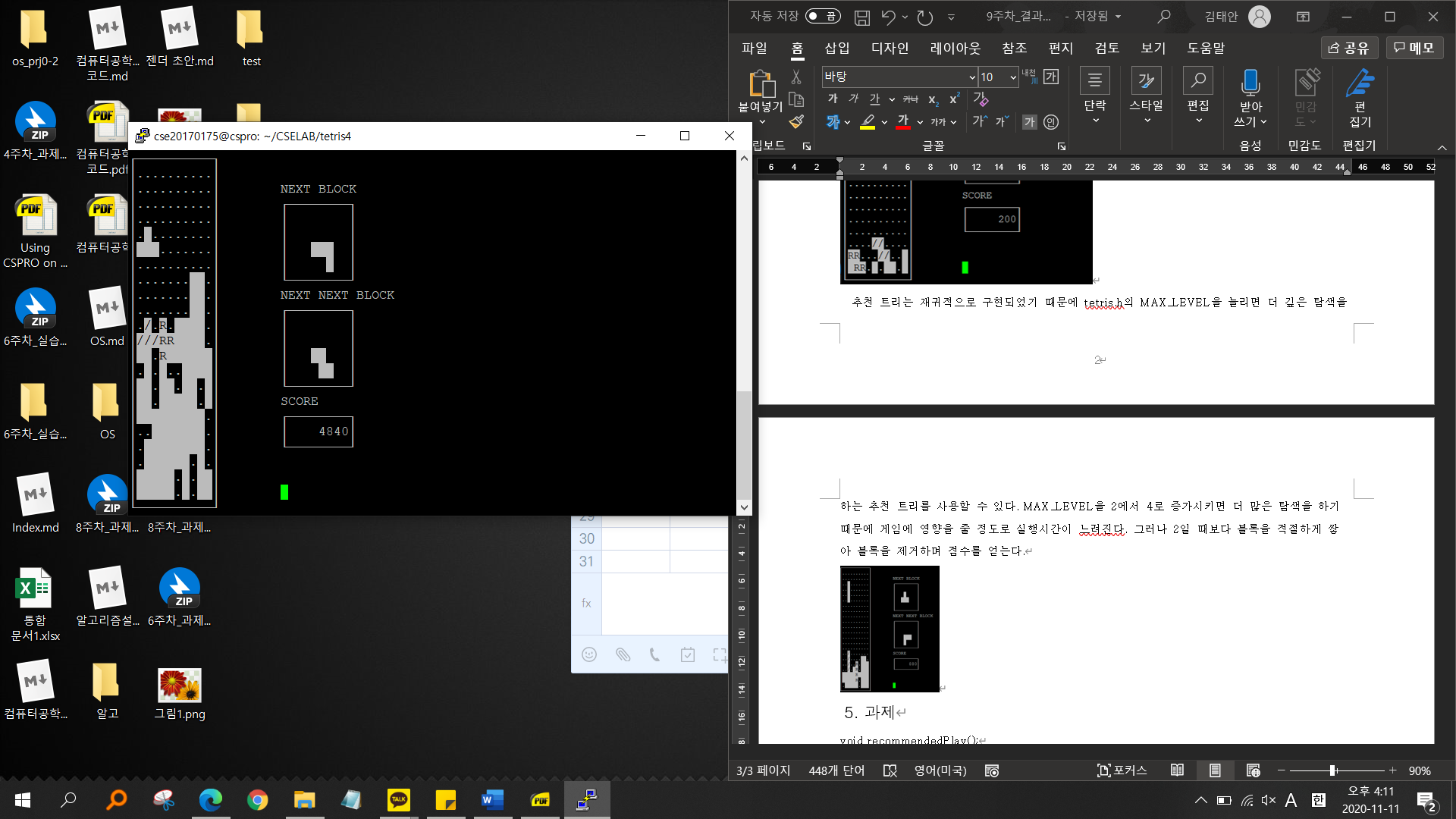
1. 실습 환경

cspro.sogang.ac.kr

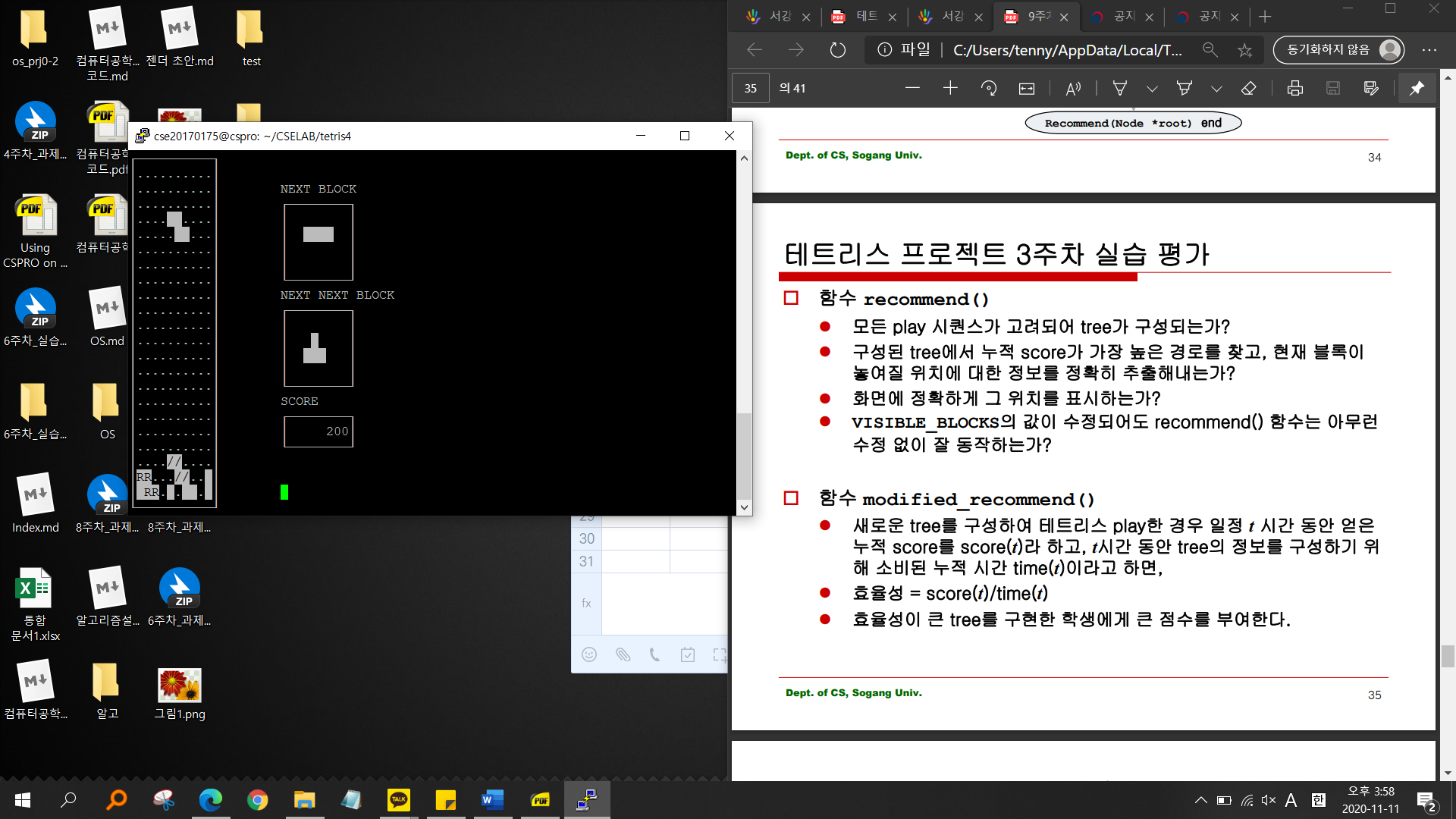
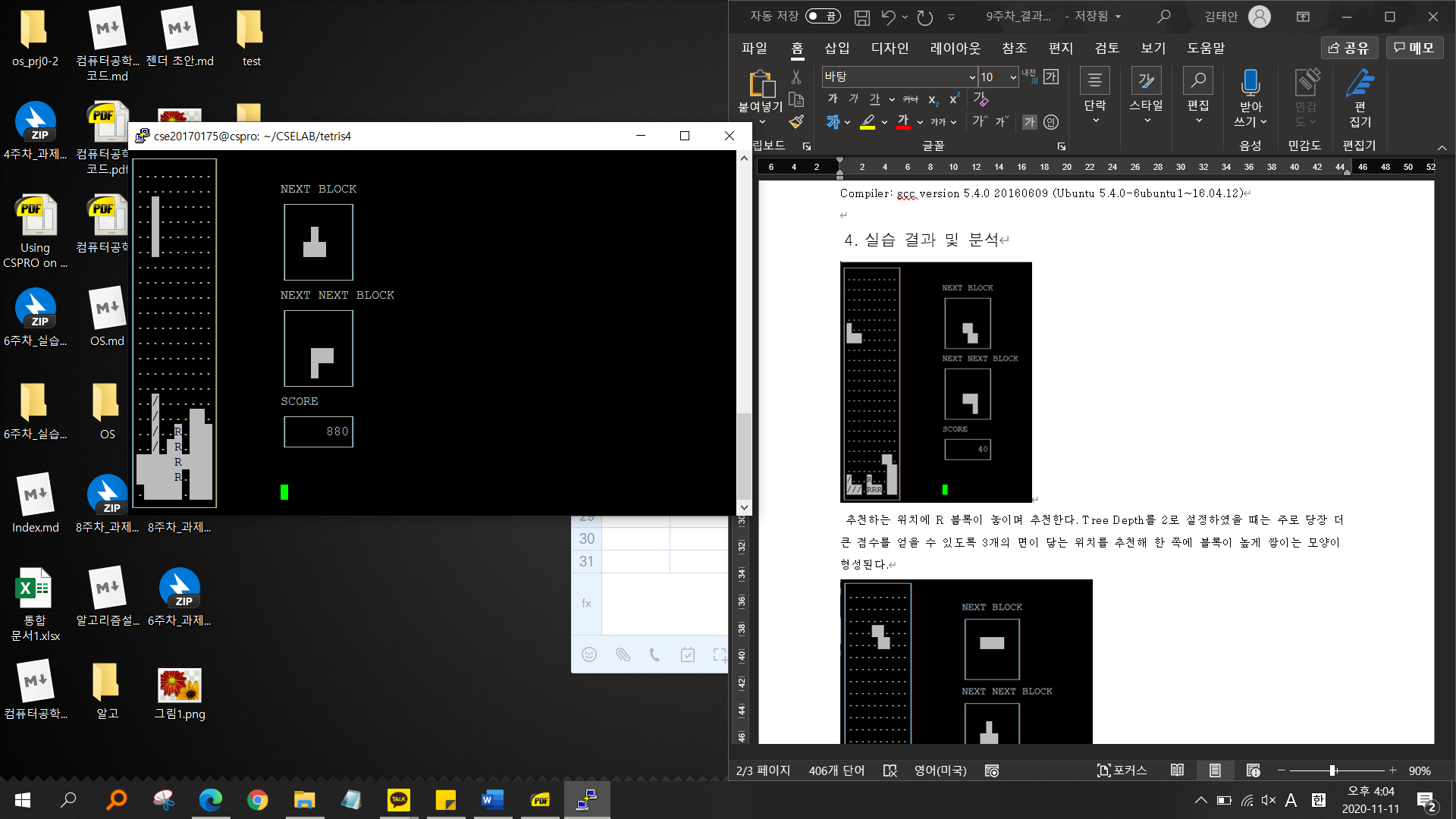
OS: Ubuntu 16.04.2 LTS (GNU/Linux 4/4/0-184-generic x86\_64)

Compiler: gcc version 5.4.0 20160609 (Ubuntu 5.4.0-6ubuntu1~16.04.12)

1. 실습 결과 및 분석

추천하는 위치에 R 블록이 놓이며 추천한다. Tree Depth를 2로 설정하였을 때는 주로 당장 더 큰 점수를 얻을 수 있도록 3개의 면이 닿는 위치를 추천해 한 쪽에 블록이 높게 쌓이는 모양이 형성된다.

추천 트리는 재귀적으로 구현되었기 때문에 tetris.h의 MAX\_LEVEL을 늘리면 더 깊은 탐색을 하는 추천 트리를 사용할 수 있다. MAX\_LEVEL을 2에서 4로 증가시키면 더 많은 탐색을 하기 때문에 게임에 영향을 줄 정도로 실행시간이 느려진다. 그러나 2일 때보다 블록을 적절하게 쌓아 블록을 제거하며 점수를 얻는다.

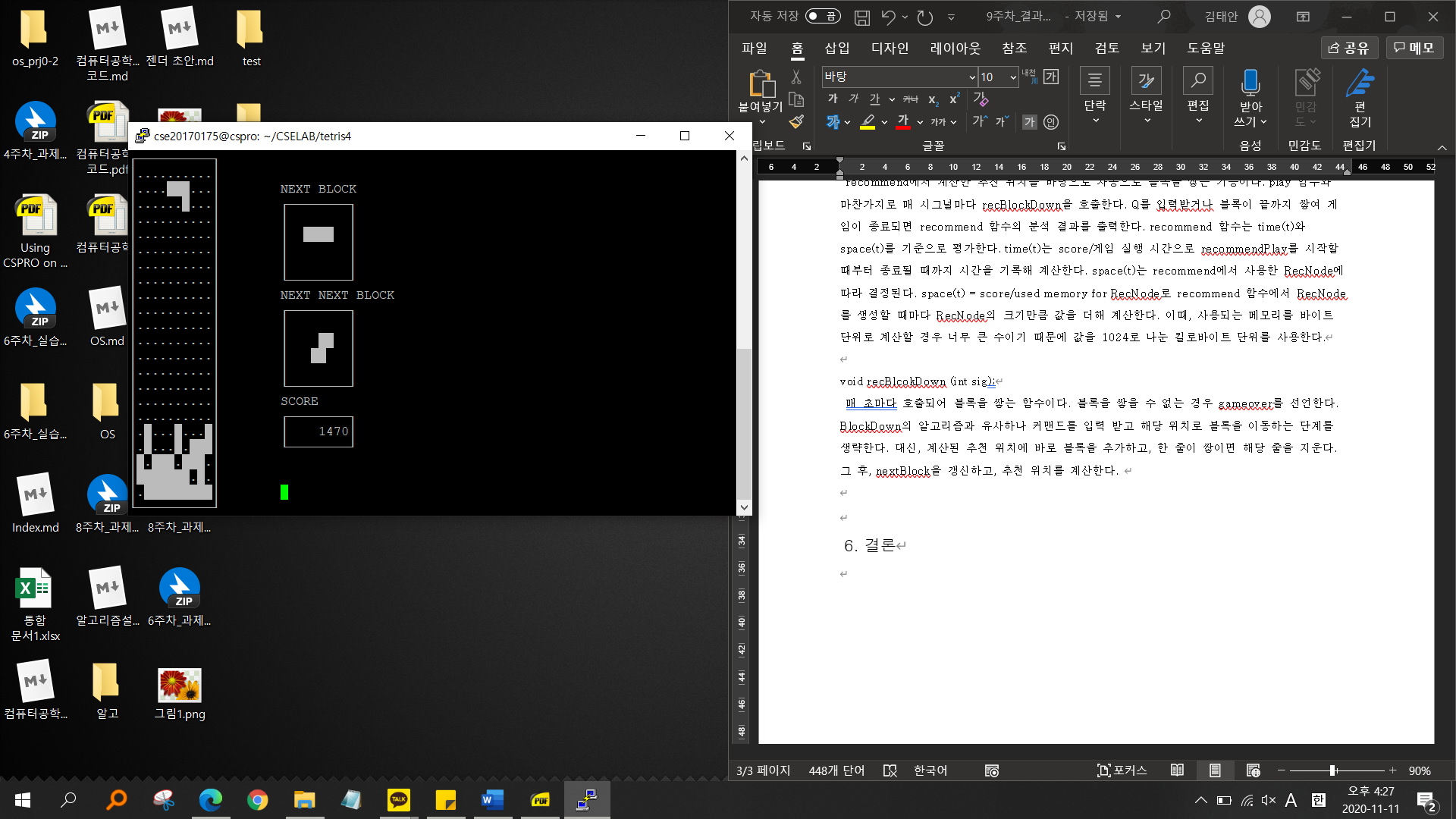
1. 과제

void recommendedPlay();

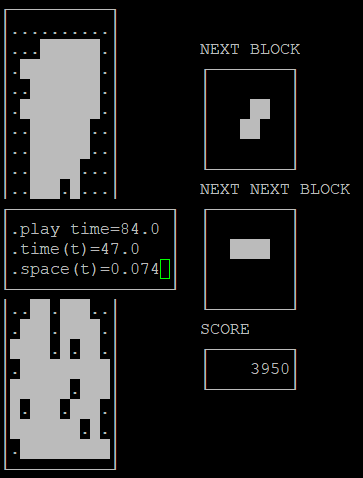
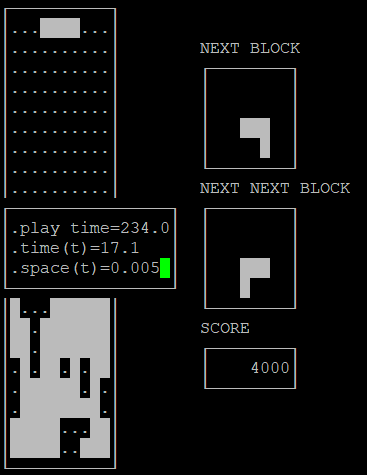
recommend에서 계산한 추천 위치를 바탕으로 자동으로 블록을 쌓는 기능이다. play 함수와 마찬가지로 매 시그널마다 recBlockDown을 호출한다. Q를 입력받거나 블록이 끝까지 쌓여 게임이 종료되면 recommend 함수의 분석 결과를 출력한다. recommend 함수는 time(t)와 space(t)를 기준으로 평가한다. time(t)는 score/게임 실행 시간으로 recommendPlay를 시작할 때부터 종료될 때까지 시간을 기록해 계산한다. space(t)는 recommend에서 사용한 RecNode에 따라 결정된다. space(t) = score/used memory for RecNode로 recommend 함수에서 RecNode를 생성할 때마다 RecNode의 크기만큼 값을 더해 계산한다. 이때, 사용되는 메모리를 바이트 단위로 계산할 경우 너무 큰 수이기 때문에 값을 1024로 나눈 킬로바이트 단위를 사용한다.

void recBlcokDown (int sig);

매 초마다 호출되어 블록을 쌓는 함수이다. 블록을 쌓을 수 없는 경우 gameover를 선언한다. BlockDown의 알고리즘과 유사하나 커맨드를 입력 받고 해당 위치로 블록을 이동하는 단계를 생략한다. 대신, 계산된 추천 위치에 바로 블록을 추가하고, 한 줄이 쌓이면 해당 줄을 지운다. 그 후, nextBlock을 갱신하고, 추천 위치를 계산한다.



recommend play 모드를 실행하면 위와 같이 블록이 자동으로 쌓이는 프로그램이 진행된다. 이를 통해 탐색 깊이에 따른 성능을 비교할 수 있다.

왼쪽은 깊이를 2, 오른쪽은 4로 설정했을 때의 결과이다. 왼쪽은 4000에 가까운 점수를 얻고 게임이 종료되었으나 오른쪽은 4000을 달성했을 때도 여유로운 모습으로, 일시정지를 통해 중단되었다. 그러나 탐색에 시간이 오래 걸리는 만큼, 플레이 시간이 3배 가까이 차이가 났으며, 시간과 공간 효율성 또한 깊이가 4일 때 매우 낮게 측정되었다. 즉, 탐색 트리의 깊이가 깊을 수록 더 나은 판단을 하나, 리소스를 효과적으로 사용하지는 못한다.

int modified\_recommend();

recommend의 실행 시간과 사용 메모리를 줄이기 위해 다음과 같은 구조를 생각해볼 수 있다. 먼저, 트리의 형태로 노드를 여러 개 추가하는 방식이 아닌, 이전보다 score가 큰 경우 값을 교체하는 방식이다. 이 경우, 시간 복잡도의 개선은 찾기 어려우나 사용하는 공간은 MAX\_LEVEL + 임시공간 1개로 매우 줄어든다.

1. 결론

이번 실습에서는 Tree를 사용해 테트리스의 추천 시스템을 구현하였다. 이때 프로그램은 모든 경우의 수를 생각하는 Brute Force 방식을 이용해 적절한 추천을 한다. 이러한 방식은 깊은 탐색을 할수록 상당히 정교한 추천을 보여주나 프로그램의 성능에 심하게 영향을 끼친다. 따라서 여러 부분에서 시간/공간의 성능을 향상 시킬 요소를 고려해보아야 한다.