2020년도 2학기 컴퓨터공학설계및실험Ⅰ

6주차 기본 테트리스 프로그램 결과 보고서

20161663 허재성

1. 실습 목적

누구에게나 친숙하고 유명한 게임인 테트리스(tetris)를 구현한다. 실험 1주차에서는 제공된 프레임(frame) 프로그램을 바탕으로, 블록의 이동, 블록의 회전, 블록을 필드에 쌓기, 줄 삭제, 점수 계산, 블록 미리 보여주기로 구성된 기본적인 기능들을 갖는 테트리스 게임을 구현하고, ncurses 라이브러리, 디버깅 방법, makefile을 만드는 방법 등을 익히도록 한다.

2. 실습 구현 내용

기본적인 테트리스 게임이 가능하도록 프레임 프로그램에 이미 구현된 함수 외에 추가적으로 구현한 함수는 다음과 같다.

(1) int CheckToMove(char f[HEIGHT][WIDTH], int currentBlock, int blockRotate, int blockY, int blockX)

현재 블록이 (blockY, blockX)로 이동할 수 있는지 확인하는 함수이다. 확인을 위해 함수 내에서 4\*4 이중 반복문(for문)을 실행한다. 반복문에서 블록 번호가 currentBlock이고, 회전 상태가 blockRotate인 블록의 모양과 블록이 이동하려는 위치인 필드의 좌표 (blockY, blockX)를 기준으로 4\*4 크기의 필드를 비교하여 블록이 필드의 좌표 (blockY, blockX)로 이동했을 때 해당 필드에 전에 쌓여 있는 블록이 있는 지 검사하여, 이미 쌓인 블록이 있을 경우 이동할 수 없으므로 거짓을 의미하는 0을, 블록이 없을 경우 참을 의미하는 1을 반환한다. 추가적으로 현재 블록이 이동했을 때, 현재 블록이 테트리스의 필드를 벗어나는 지 검사하여 벗어날 경우 0을 반환한다.

input: 필드 정보를 나타내는 2차원 문자형 배열 f[HEIGHT][WIDTH], 이동하려는 현재 블록의 번호를 나타내는 정수형 변수 currentBlock(0~6), 현재 블록의 회전상태를 나타내는 정수형 변수, blockRotate(0~3), 현재 블록이 이동하려는 필드 위치의 y좌표 정수형 blockY, x좌표 정수형 blockX

output: 현재 블록이 이동하려는 좌표 (blockY, blockX)로 이동 가능할 경우 1을 반환, 이동 불가능할 경우 0을 반환

(2) void DrawChange(char f[HEIGHT][WIDTH], int command, int currentBlock, int blockRotate, int blockY, int blockX)

­­현재 이동한 블록이 필드의 위치 (blockY, blockX)에 있을 때 입력받은 이동 명령키(아래로 이동하는 아래 방향키, 좌우로 이동하는 좌우 방향키, 블록을 회전시키는 위 방향키)에 따라 이동 전 블록이 필드 상에 놓인 위치와 블록의 회전 상태를 찾고, 필드의 해당 위치에 그려져 있는 블록을 지운다. 이전 위치의 블록을 지운 후 새로 이동한 위치인 (blockY, blockX)를 기준으로 현재 블록의 번호 currentBlock, 현재 블록의 회전 상태인 blockRotate를 고려하여 필드에 이동 후의 블록을 그린다. 이동을 마친 블록을 그리기 위해 이미 정의된 DrawBlock 함수를 호출한다. 이전 블록의 위치를 알아내기 위해, 이전 블록의 좌표를 나타내는 변수 prevY, prevX와 이전 블록의 회전 상태를 나타내는 prevR을 선언하고 각각 blockY, blockX, blockRotate로 초기화한다. switch문을 사용해 입력받은 명령어 command에 따라, 이전 위치를 유추한다. 예를 들어 입력받은 command 키가 회전을 의미하는 위 방향키였을 경우, prevR 상태에서 한 번 더 회전하여 prevR->blockRotate: 0->1, 1->2, 2->3, 3->0으로 변화하였으므로, prevR은 blockRotate->prevR: 0->3, 1->0, 2->1, 3->2가 된다. 즉

|  |
| --- |
| prevR = (blockRotate + 3) % 4; |

로 표현 가능하다. 이전 블록의 위치와 회전 상태를 알아낸 후, 4\*4 이중 for문을 실행해 해당 위치에 대응하는 block 배열의 값이 1일 경우, 필드의 해당 칸에 ‘.’을 출력하여 이전 블록이 지워졌음을 구현한다.

input: 필드 정보를 나타내는 2차원 문자형 배열 f[HEIGHT][WIDTH], 입력받은 명령어를 나타내는정수형 변수 command, 이동하려는 현재 블록의 번호를 나타내는 정수형 변수 currentBlock(0~6), 현재 블록의 회전상태를 나타내는 정수형 변수, blockRotate(0~3), 현재 블록의 필드 위치의 y좌표 정수형 blockY, x좌표 정수형 blockX

output: 없음

(3) void AddBlockToField(char f[HEIGHT][WIDTH], int currentBlock, int blockRotate, int blockY, int blockX)

현재 블록이 완전히 내려와 필드에 추가될 때 호출되는 함수이다. 4\*4 이중 for문을 돌아서 현재 블록의 번호와, 회전 상태를 고려 후, 대응되는 위치의 필드 값을 1로 바꾸어 블록이 필드에 추가되었음을 구현한다. 마지막으로 DrawField 함수를 호출하여 블록이 필드에 추가된 상황을 그린다.

input: 필드 정보를 나타내는 2차원 문자형 배열 f[HEIGHT][WIDTH], 이동하려는 현재 블록의 번호를 나타내는 정수형 변수 currentBlock(0~6), 현재 블록의 회전상태를 나타내는 정수형 변수, blockRotate(0~3), 블록이 필드에 추가되는 위치의 y좌표 정수형 blockY, x좌표 정수형 blockX

output: 없음

(4) int DeleteLine(char f[HEIGHT][WIDTH])

테트리스 게임에서 블록이 끝까지 내려와 필드에 추가된 후 호출되는 함수이다. 블록이 추가된 후 필드를 아래 방향으로 행 별로 탐색하여 꽉 찬 행이 있는지 탐색한다. 꽉 찬 행이 있으면 해당 행을 지우고 지워진 행으로 위의 행의 필드 값을 한 칸씩 내린다. 필드를 탐색하기 위해 이중 for문을 이용한다. 행의 크기가 HEIGHT이고, 열의 크기는 WIDTH이다. 바깥쪽 loop는 i=0부터 i=HEIGHT-1까지 총 HEIGHT번 실행되며, 안쪽 loop는 j=0부터 시작해서 j=WIDTH-1까지 총 WIDTH번 실행된다. 안쪽 loop에서 만약 필드의 값 f[i][j]가 0일 경우, 해당 행(줄)이 꽉 차지 않았음을 의미하므로 해당 행에 대해서 탐색을 멈추고 다음 행을 탐색한다. 행 i에서 j=0~WIDTH-1까지 모든 f[i][j]가 1일 경우, 지워질 줄의 수를 의미하는 정수형 변수 line의 값을 1 증가시킨다(초기값은 0). 지워질 줄이 있을 경우, 지워질 줄 i의 윗줄인 i-1부터 가장 윗 줄인 -1행까지 모든 필드 값을 반복문을 이용하여 1칸씩 아래로 내린다. 가장 윗 줄의 경우 내릴 값이 없으므로 0으로 설정한다. 지워질 줄의 수 line을 구한 후, 점수 계산 법에 따라 line \* line2 \* 100을 리턴한다.

DeleteLine 함수에서 line을 찾기 위한 이중 for문의 시간복잡도는 O(HEIGHT\*WIDTH)이다. 지워질 줄이 있을 경우, 지워질 i번째 줄을 찾은 후, i번째 줄의 윗줄들을 한 칸씩 내리는 반복문의 시간복잡도는 O(i\*WIDTH)이다. 최악의 경우는 테트리스 필드의 가장 아랫줄이 지워질 경우, 즉 i=HEIGHT-1일 때이고 이 때의 시간 복잡도는 O(HEIGHT\*WIDTH)이다.

테트리스 게임 규칙 상 한 번에 지워질 수 있는 줄의 수는 최대 4줄이다. 따라서 line의 값도 최대 4이다. 즉 시간 복잡도는 최대 O(4\*HEIGTH\*WIDTH)이다. 상수의 곱은 무시할 수 있으므로, 시간 복잡도는 최대 O(HEIGHT\*WIDTH)이다.

공간 복잡도의 경우, 테트리스 필드를 위한 공간 복잡도는 O(HEIGTH\*WIDTH)이다.

input: 필드 정보를 나타내는 2차원 문자형 배열 f[HEIGHT][WIDTH]

output: 지워지는 줄 수인 line에 따른 점수 증가량 line\*line\*100

(5) void BlockDown(int sig)

현재 블록을 한 칸 아래로 이동시키는 함수이다. 한 칸 아래로 이동 가능한 지 확인하기 위해 위에서 정의한 CheckToMove 함수를 호출한다. CheckToMove 함수를 호출할 때, 한 칸 아래로 이동이 가능한 지를 확인해야 하므로 이동한 칸의 Y 좌표로 blockY + 1을 인자로 전달해 준다. CheckToMove 함수의 반환 값이 1일 경우, 아래로 한 칸 이동이 가능하므로 blockY의 값을 1 증가시키고, 위에서 정의한 DrawChange 함수를 호출하여 한 칸 이동한 후의 상황을 필드에 그린다.

CheckToMove 함수의 반환 값이 0일 경우, 더 이상 아래로 내릴 수 없다. 더 이상 아래로 내릴 수 없는 상황에서 현재 블록 위치의 Y 좌표인 blockY가 -1일 경우, 블록이 초기 위치에서 더 이상 내려올 수 없다는 의미이므로 gameOver를 1로 바꾸어 테트리스 게임이 종료했음을 알린다. 더 이상 아래로 갈 수 없으므로 blockY의 값과 관련 없이 위에서 정의한 AddBlockToField 함수를 호출하여 현재 블록이 필드에 추가되도록 한다. 블록이 필드에 추가된 후, 지워진 줄 수를 계산하여 그에 맞게 점수를 갱신하기 위해 DeleteLine 함수를 호출한다.

현재 블록이 완전히 내려가서 테트리스 필드에 추가되었으므로 다음 블록을 나타내는 nextBlock[1]이 현재 블록이 되도록 한다. 다음 블록을 현재 블록으로 바꾸고, 그 다음 블록을 rand 함수를 호출하여 정하고 이미 정의된 DrawNextBlock 함수를 호출하여 다음 블록 정보를 화면에 그린다. 새로운 현재 블록의 위치를 테트리스 필드의 상단, 중앙인 (-1, WIDTH/2 – 2)에 배치시키기 위해 blockY, blockX를 각각 -1, WIDTH/2-2로 초기화시키고 blockRotate도 0으로 한다. 갱신된 점수를 출력하기 위해 이미 정의된 PrintScore 함수를 호출하고 DrawField 함수를 호출하여 새로운 현재 블록을 필드에 그린다. BlockDown이 끝났음을 알리기 위해 timed\_out을 1로 설정한다.

BlockDown 함수의 input을 이해하기 위해 play 함수를 관찰하면 act.sa\_handler = BlockDown; 이라는 문장이 있다. 신호가 오면 BlockDown 함수를 실행한다는 의미이다. BlockDown 함수의 input인 sig는 해당 신호를 의미한다.

input : 신호에 해당하는 정수형 변수 sig

output: 없음

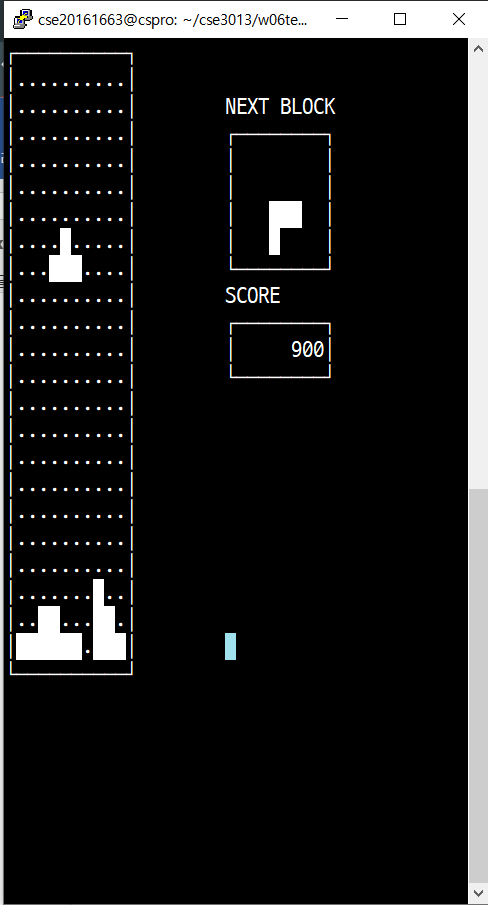
3. 실습 환경

CSPRO 서버 : CSPRO.sogang.ac.kr

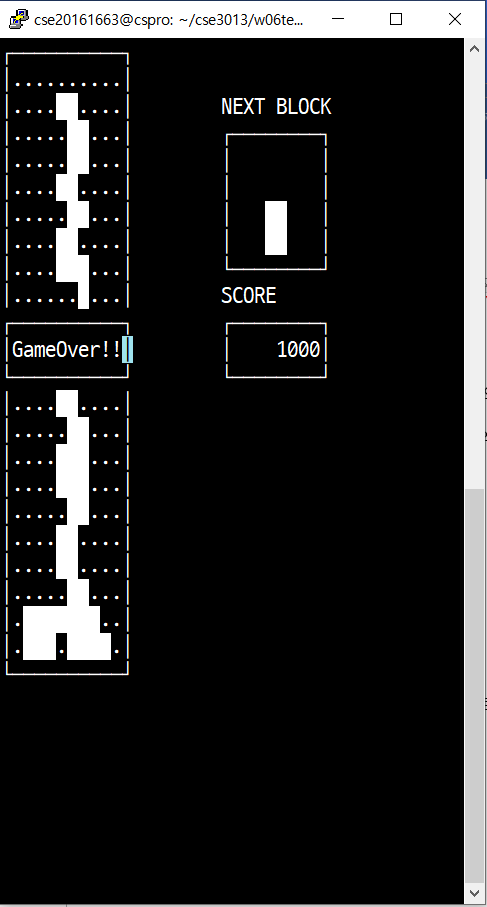
4. 실습 결과 및 분석



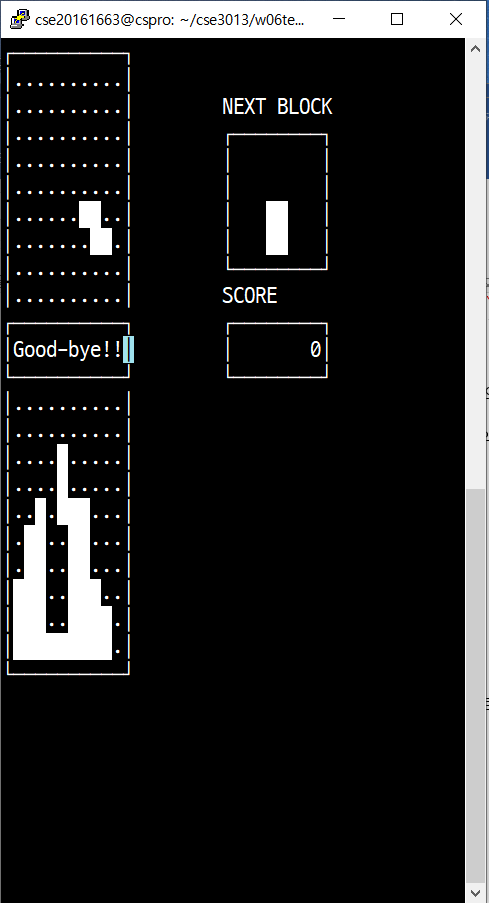
게임을 시작 후 블록이 어느 정도 쌓인 모습이다. 블록이 잘 쌓이는 것을 알 수 있다. 필드에서 현재 블록이 내려오는 모습과, 다음 블록을 화면 오른쪽에서 확인할 수 있다.



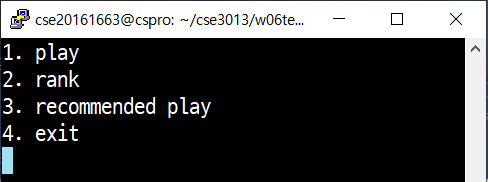
위의 상황에서 블록을 끝까지 내려 세 줄이 지워진 후의 상황이다. 점수 계산법에 따라 900점이 오른 것을 확인할 수 있으며, 위의 캡처 화면에서의 다음 블록이 현재 블록이 된 것도 확인할 수 있고, 새로운 다음 블록이 임의로 정해진 것도 확인할 수 있다.



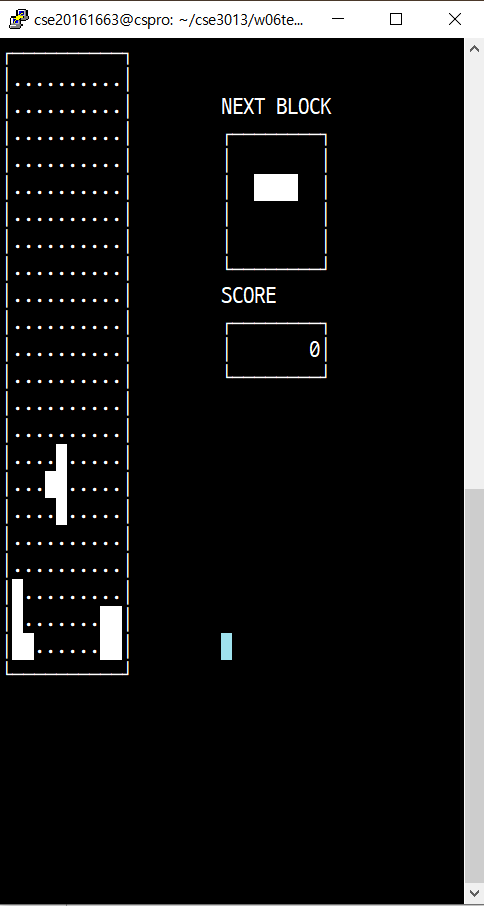
블록이 쌓여서 더 이상 새로운 블록이 내려갈 수 없을 경우, GameOver!!메시지가 뜬다.



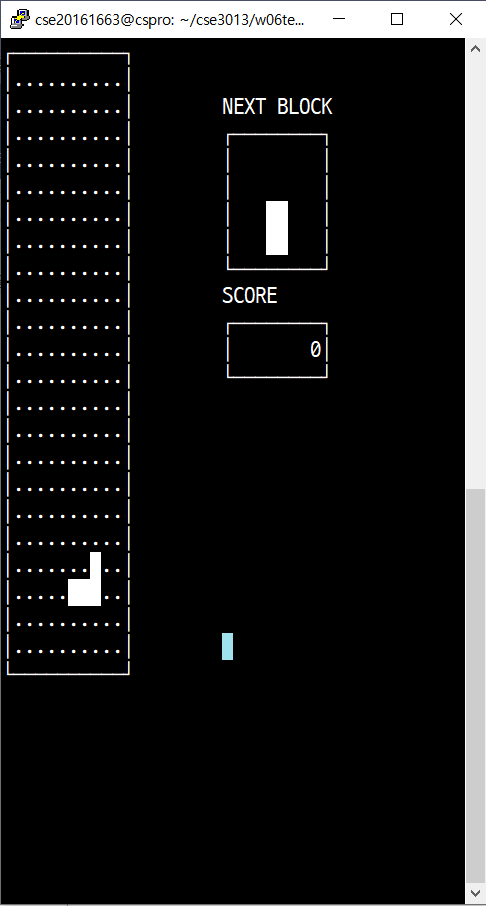
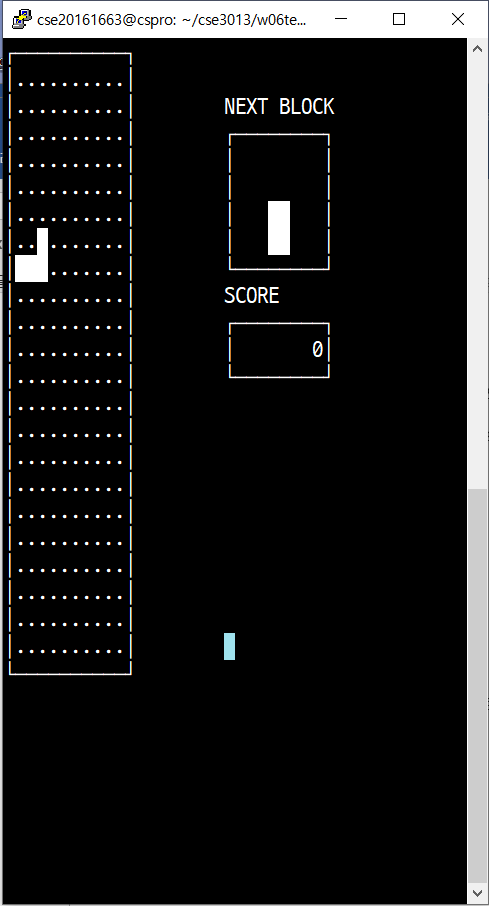
게임 진행 중, 알파벳 Q를 누를 경우, Good-bye!! 메시지가 뜨며 게임이 강제로 종료된다. 어떠한 방법으로든 게임이 종료 된 후, 아무 키나 누르면 메뉴 화면으로 돌아온다.



메뉴 화면에서 1을 입력할 경우 테트리스 게임이 시작되고, 4를 입력할 경우 테트리스 프로그램이 종료된다. 2와 3은 아직 구현되지 않았다.



블록이 잘 회전하는 것을 확인할 수 있다. 아래 방향키를 누르지 않아도 자동으로 일정 시간마다아래로 내려가는 것 또한 확인 가능하다.



블록이 좌우로 잘 움직이는 것도 확인 가능하다. 필드를 벗어나지 않는 것도 확인할 수 있다.

하나의 블록이 내려오는 시간은 테트리스 필드의 세로 길이에 비례하므로 O(HEIGHT)의 시간 복잡도를 가진다.

5. 과제

6주차 과제로 현재 블록의 그림자를 구현한다. 또한 기존의 점수 계산식에 추가적으로 현재 블록이 필드에 추가될 경우 블록이 놓여진 위치에서 필드와 블록의 아래 부분이 맞닿은 면적의 수에 비례하여 점수를 증가시키는 방법을 추가한다. 즉 새로운 점수 계산 방식은 다음과 같다.

|  |
| --- |
| 점수 = (지워진 줄 수)2 x 100 + (필드와 블록의 아래 부분이 맞닿은 면적의 수) x 10 |

마지막으로 기존의 테트리스 게임에서는 다음 블록을 1개의 상자에서 1개만 보여줬다. 1개의 상자를 더 추가해서 다음 다음 블록의 정보를 보여주도록 한다.

위의 세 가지를 구현하기 위해 기존의 함수들을 수정하고 새로운 함수들을 정의해야 한다.

수정한 함수

(1) void InitTetris()

기존에는 다음 블록인 nextBlock[1]까지만 rand 함수를 호출해서 초기화했지만, 수정 후 nextBlock[2]까지 rand 함수를 호출해 초기화한다. 이를 위해서 tetris.h 헤더 파일에 선언된 nextBlock 배열의 크기인 BLOCK\_NUM을 2에서 3으로 수정해야 한다. 또한 함수 내에서 호출했던 DrawBlock 함수를 뒤에서 설명할, 새로 정의한 함수 DrawBlockWithFeatures를 호출하는 것으로 대신한다.

(2) void DrawOutLine()

기존에는 다음 블록을 보여주는 상자를 1개만 그렸지만 2개를 보여주게 수정한다. 알맞은 위치을 정하고 DrawBox 함수를 호출하여 새로운 상자를 추가적으로 그린다. 상자가 한 개 추가되어 점수를 보여줄 상자의 위치가 변경되어야 하므로 이를 반영한다.

(3) void PrintScore(int score)

기존과 점수 상자의 위치가 달라졌으므로 move 함수의 인자를 조정해 점수가 출력될 위치를 변경한다.

(4) void DrawNextBlock(int \*nextBlock)

그려야 할 다음 블록 상자가 하나 증가했으므로 추가적으로 그려준다.

(5) void DrawChange(char f[HEIGTH][WIDTH]. int command, int currentBlock, int blockRotate, int blockY, int blockX)

기존에는 이전 블록의 정보를 지우고, DrawBlock 함수를 호출해 이동한 위치에 새롭게 그렸다. 여기에 추가하여 이전 블록의 그림자 위치를 찾아, 이전 블록의 정보를 지울 때와 마찬가지 방법으로 그림자 정보를 지운다. 그리고 DrawBlock 함수 대신 뒤에서 설명할 DrawBlockWithFeatures 함수를 호출한다. 그림자 정보를 찾는 방법은 뒤에서 정의할 DrawShadow 함수에서 설명한다.

(6) int AddBlockToField(char f[HEIGHT][WIDTH], int currentBlock, int blockRotate, int blocky, int blockX)

원래 함수는 반환형이 void였지만 수정된 함수에서는 맞닿은 부분에 비례한 점수를 계산하여 반환한다.

현재 블록이 필드에 추가될 때, 현재 블록의 아래 부분이 필드와 맞닿은 면적을 계산하여 변수 touched에 저장하는 과정을 추가한다. (touched의 초기값은 0)

touched를 계산하기 위해 4\*4 이중 for문을 돌아서 block[currentBlock][blockRotate][i][j]==1이고 필드에 대응하는 위치의 y좌표 blockY+i+1가 HEIGHT이거나(즉 필드 바닥에 맞닿거나), f[blockY+i+1][blockX+j]==1일 경우(이미 채워진 부분에 맞닿을 경우) touched가 증가하도록 한다. 변수 Score에 touched\*10의 값을 저장 후 반환한다.

(7) void BlockDown(int sig)

기존의 경우, 새로운 블록의 정보가 1개였으나 수정 후 2개로 증가하였으므로 nextBlock[0]에 nextBlock[1]을 대입 후, nextBlock[1]에 nextBlock[2]를 대입한 후, rand 함수를 호출해 nextBlock[2]의 값을 새로 정한다. 또한 기존의 점수 계산 방식에 추가적인 점수 계산 방식을 고려하여 전역 변수 score에 AddBlockToField 함수의 반환값을 추가로 더해준다.

새로 정의한 함수

(1) void DrawShadow(int y, int x, int blockID, int blockRotate)

현재 블록의 모양(blockID : 0~6), 회전 상태(blockRotate: 0~3)을 고려해 현재 블록이 최대로 내려갔을 때의 위치를 파악 후 해당 위치에 그림자를 그린다. 최대로 내려간 위치를 파악하기 위해 그림자의 y좌표가 될 변수 shadowY를 선언 후, 현재 위치 y로 초기화한다. shadowY를 1씩 증가시켜가며 CheckToMove 함수를 반복적으로 호출한다. CheckToMove 함수가 0을 반환하면 반복을 종료하고 그 떄의 shadowY와 x를 기준으로 그림자를 그린다. 그림자를 그리기 위해서 DrawBlock 함수를 호출하는데 기존의 블록을 공백으로 출력했지만, 그림자는 ‘/’으로 출력하므로 인자로 ‘ ‘ 대신 ‘/’를 전달한다.

최대로 내려갈 수 있는 거리는 필드의 세로 값인 HEIGHT에 비례하므로 시간 복잡도는 O(HEIGHT)이다.

input: 그림자를 그릴 현재 블록의 좌표 (y, x)의 좌표 y, x, 현재 블록 번호 blockID(0~6), 현재 블록의 회전 상태 blockRotate(0~3)

output: 없음

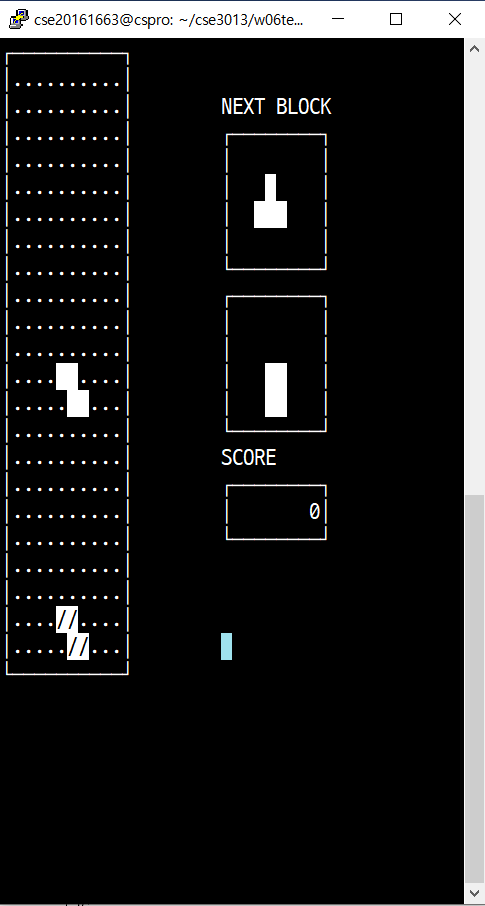
(2) void DrawBlockWithFeatures(int y, iny x, int blockID, int blockRotate)

기존의 DrawBlock 함수를 확장한 함수로 현재 블록의 위치 (y, x)와 ,블록 번호 blockID, 회전 상태 blockRotate를 기준으로 블록의 그림자와 현재 블록을 모두 그리는 함수이다. DrawShadow 함수와 DrawBlock 함수를 모두 호출한다.

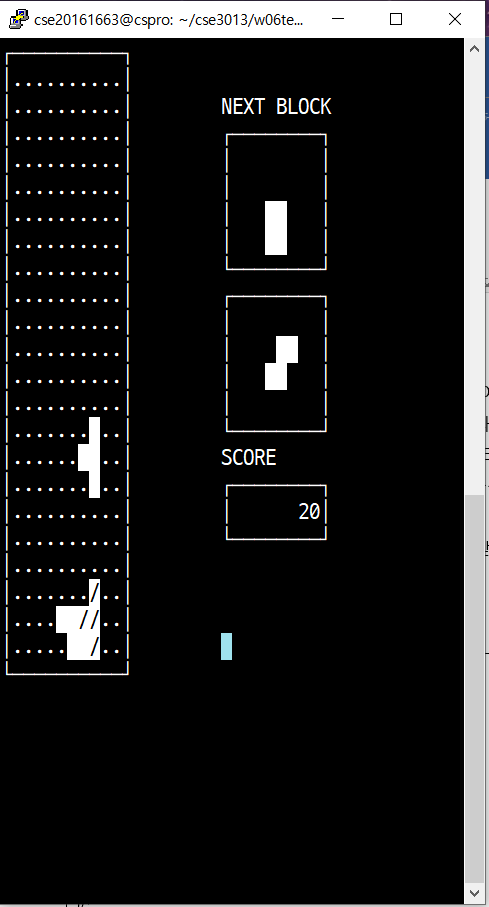
input: 그림자를 그릴 현재 블록의 좌표 (y, x)의 좌표 y, x, 현재 블록 번호 blockID(0~6), 현재 블록의 회전 상태 blockRotate(0~3)

output: 없음

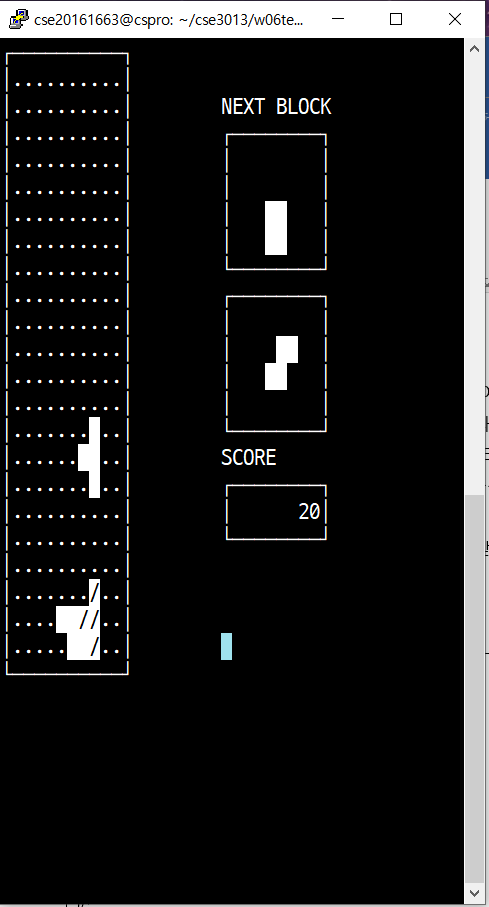
구현한 결과이다.



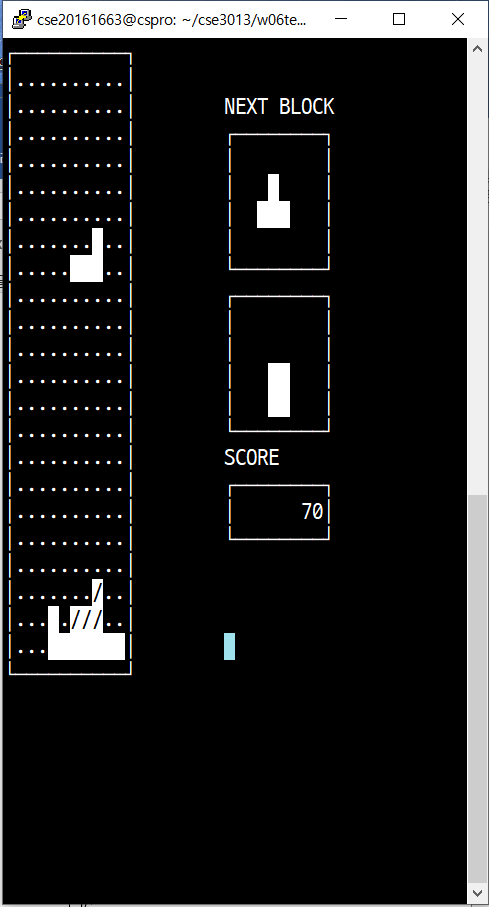
현재 블록과 현재 블록의 그림자가 모두 잘 나타난다. 또한 화면 오른쪽에 NEXT BLOCK 정보를 2개 보여주고 있다.



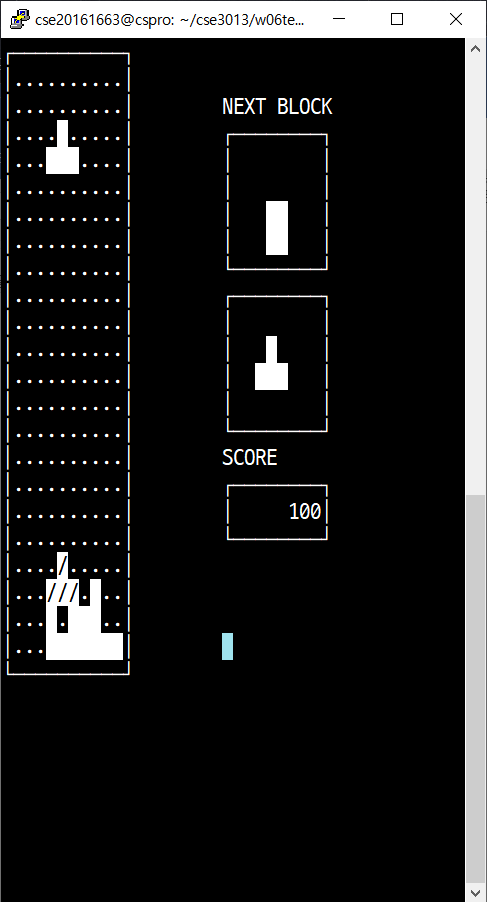
첫 번째 블록이 떨어지고 점수가 20점 증가한 것을 확인할 수 있다. 바닥과 맞닿은 면적이 2칸이므로 점수는 2\*10=20점이 증가했다. 블록이 회전한 상태인데, 그림자도 같이 회전한 것이 확인 가능하다.



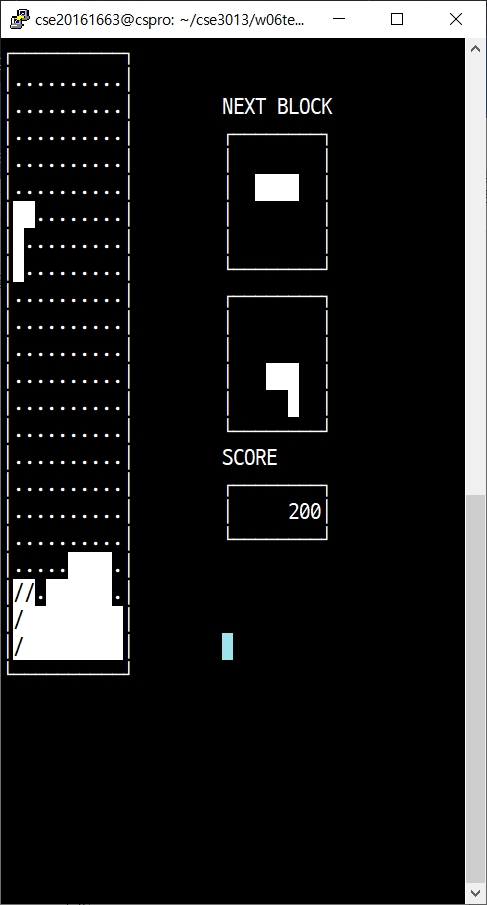
바닥과 첫 번째 블록이 맞닿은 면적이 2칸이므로 20점이 증가하였다.



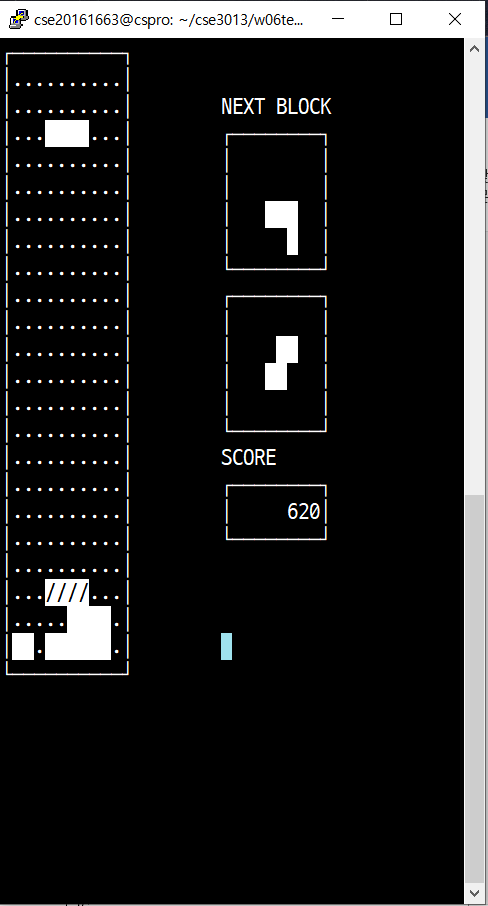
바닥이 아닌 이미 쌓인 블록 위에 새로운 블록을 놓아 본다.



이미 쌓인 블록 위에 세 칸이 맞닿아 점수가 30점이 증가했다.



바닥에 한 칸, 블록에 한 칸이 맞닿고 추가적으로 2줄이 지워질 상황을 본다.



두 줄이 지워져서 22x100=400점이 증가했고 추가적으로 바닥 1칸, 이미 쌓인 블록 1칸이 맞닿아 2x10=20점이 추가로 증가해 총 420점이 증가하였다.

위의 경우를 관찰해보면 첫 번째 다음 블록과 두 번째 다음 블록도 랜덤하게 잘 나타나는 것을 알 수 있다.

기타

6주차 테트리스 프로그램에서 시간, 공간 복잡도에 영향을 주는 요소는 HEIGHT와 WIDTH뿐이다. 여러 함수에서 이중 for문으로 4\*4 반복문을 돌았지만 모두 O(1)의 시간 복잡도를 가진다.