C++ 프로그래밍 설계 과제 : 테트리스 만들기 2단계

20132926 전호현

요 약

본 과제에서는 설계과제 #1에서 나아가 한 모양의 테트로미노 블록으로만 진행되는 테트리스 게임을 만든다.
본 과제의 목표는 오직 O-블록만으로 진행되는 테트리스 게임을 구현하는 것이다.

---

1. Tetris.cpp , Tetris.h

* 파란색 함수 또는 클래스는 본 파일의 뒷부분에서 정의한다

테트리스 클래스에서는 사용자로부터 키보드 값을 입력받아 그것을 화면에 출력하는 역할을 수행해야 한다. 다시말해서, BoardPane의 draw를 수정해 주어야 한다. 일단, 사용자로부터 키보드 값을 입력받기 위해 Tetris의 생성자에 keypad(stdscr, TRUE) 와 noecho() 함수를 호출한다.

- BoardPane 내부에 값을 넣어주기 위해 Tetris.h에 Board b를 선언한다. 여기서 Board클래스는 뒤에서 정의하기로 한다. 그리고 Board b의 가로와 세로 길이를 나타낼 정적 상수 변수를 선언한다.
  static const int B_WIDTH = 18; static const int B_HEIGHT = 20;

- 테트리스의 생성자에서 Board b를 초기화한다. b(B_WIDTH, B_HEIGHT);

- Tetris::play()

  다음으로 테트리스의 play() 함수를 수정한다. play() 함수에서는 사용자로부터 입력받을 input 변수, 게임 오버인지 아닌지를 판별할 bool형 변수 gameOver와 새로운 블록이 나와야 하는지 아닌지를 판별할 bool 형 변수 newBlock을 선언한다. updateScreen() 함수를 통해 화면을 업데이트 하고, 사용자로부터 input값을 하나 입력받아 게임 시작을 인지한다. 게임이 시작되면, While(1) 반복문을 만들어 gameOver가 false일 경우에 계속해 반복문을 수행하도록 한다. 반복문 내부에서는 Board클래스의 eraseBlock()을 호출하여 O블록으로 채워진 부분들을 모두 지워준다. 그리고 gameOver에 Board클래스의 insertBlock()을 호출한 결과값을 반전하여 넣어준다. insertBlock()은 블록을 넣을 부분이 있을 경우 true를, 없을 경우 false를 반환한다. 만일 여기서 gameOver 값이 true가 되었다면, 사용자로부터 입력값을 하나 받고 반복문을 종료시킨다.

  만일 gameOver가 false여서 반복문이 종료되지 않았다면, 또 다른 반복문을 새로 수행한다. 이 반복문은 newBlock이 false일 경우에만 반복문을 수행한다. 이 반복문에서는 사용자로부터 키보드 값을 입력받아 그 케이스에 따라 다른 함수를 호출한다. 왼쪽 방향키일 경우, Board 클래스의 moveBlockLeft(), 오른쪽 방향키일 경우 moveBlockRight(), 아래 방향키의 경우 moveBlockDown(), 스페이스 바의 경우 dropBlock()을 호출한다. moveBlockLeft(), Right(), Down()은 모두 bool형을 반환하는데, 이 값이 false일 경우 newBlock변수 값이 true가 되어 반복문을 종료한다. 자세한 함수의 구현은 뒷부분에서 설명한다. dropBlock()의 경우에는 void형의 함수이므로 함수를 수행한 후에 반복문이 종료 될 수 있도록 newBlock값을 true로 바꿔준다. 함수 호출 후 변경된 사항이 반영될 수 있도록 updateScreen()을 호출한다.

  결론적으로, while(1) 반복문은 gameOver가 아닐 경우 계속해서 새로운 블록을 생성해주는 반복문이고, while(!newBlock) 반복문은 입력받은 방향키에 따라 블록을 옮기고 블록이 더 이상 아래로 내려갈 수 없는 경우에 반복문 수행을 종료하는 반복문이다.

- Tetris::updateScreen();

  updateScreen() 함수에서는 각 Pane들이 Board b값을 토대로 draw()를 수행할 수 있도록 draw의 매개변수로 b를 넘겨준다. 이번 과제에서는 InfoPane, BoardPane, StatPane 값에만 변화를 줄 것이므로 세 개의 draw에만 b를 넘긴다.

2. Board.h, Board.cpp

Board 클래스는 블록을 그릴 2차원 배열을 제공하고, 배열 내부에 블록을 삽입하거나 좌, 우, 아래, 드롭 등을 통해 블록을 움직이는 함수를 제공한다. 그리고 꽉 찬 보드의 행을 지워주는 함수도 제공한다. 일반적인 테트리스의 경우에는 테트로미노에 따라 다른 움직임을 제공하도록 Block을 매개변수로 받아주어야 하지만, 이번 과제에서는 어차피 O블록만 이용하므로 Block은 따로 구현하지 않고 Board 클래스에 그대로 O 블록의 움직임을 적어 구현하였다.
먼저, 헤더파일의 자세한 구현은 아래와 같다.

```
#ifndef _BOARD_H_
#define _BOARD_H_

//#include "Block.h"

enum Cell{VOID, I, O, L, J, S, Z, T}; // 배열 내부에 들어갈 값들을 enum으로 정의.
                                      각각은 테트로미노 모양을 뜻한다.

class Board{

        // Board클래스의 private 변수들을 사용할 클래스들을 friend 클래스로 지정한다.
        friend class StatPane;
        friend class InfoPane;
        friend class BoardPane;
        friend class Block;(현 과제에서는 사용하지 않는다)

private:
        int height; // 배열의 높이
        int width; // 배열의 너비
        Cell **cells; // 블록을 그릴 2차원배열
        int t_x, t_y; // 배열 내부에서 현재 x좌표와 y좌표를 나타내줄 변수
        Cell curBlock; // 현재 블록을 나타내는 변수(현 과제에서는 사용하지 않는다)
        int fulllines; // 꽉 찬 라인의 개수를 세는 변수
        int blockcount; // 현재까지 나온 block의 개수를 새는 변수

public:
        Board(int w, int h); // 생성자
        ~Board(); // 소멸자
        bool insertBlock(); // 블록을 삽입하는 함수
        bool moveBlockLeft(); // 블록을 왼쪽으로 이동시키는 함수
        bool moveBlockRight(); // 블록을 오른쪽으로 이동시키는 함수
        bool moveBlockDown(); // 블록을 아래로 이동시키는 함수
        void dropBlock(); // 블록을 맨 아래로 떨어뜨리는 함수
        void eraseBlock(); // 꽉 찬 행을 지워주는 함수
};

#endif
```

Board 클래스의 중요 함수들만 설명하면,

- Board::Board(int w, int h)

  Board의 생성자의 주요 역할은 BoardPane 내부에 그려질 2차원 배열 cells를 초기화하는 것이다. 매개변수로 받은 w와 h를 width와 height에 넣어주고, cells배열을 이 값만큼 동적으로 할당한다. 그리고 모든 값들을 VOID로 초기화한다. 여기서, 가로와 세로의 비율이 2:1 임을 생각하여 가로 cell은 w/2만큼만 동적할당한다. 나머지 int 변수들을 모두 0으로 초기화해준다.

- bool Board::insertBlock()
  블록을 삽입하는 역할을 하는 함수이다. 먼저, 지금까지 내려온 블록의 개수를 반환하기 위해 blockcount를 1 증가시킨다. 그리고 BoardPane의 가운데 좌표를 초기 t_y, t_x 값으로 지정한다. 이 값의 기준은 O-블록의 왼쪽 아래 값으로 한다. 그리고 if문을 통해 cells[t_y][t_x] 값이 VOID일 경우에만 O-블록의 모양을 cells배열에 지정하고, true 값을 반환한다. 만일 이 값이 VOID가 아닐 경우에는 새로운 블록이 들어갈 자리가 없다는 것을 뜻하므로 false를 반환한다.

- bool Board::moveBlockLeft()
  블록의 좌표값을 왼쪽으로 이동시키는 역할을 하는 함수이다. 초기 t_y, t_x의 기준값은 O-블록의 왼쪽 아래 이다. 만일 현재 좌표에서 O-블록의 왼쪽 부분의 두 칸이 모두 VOID 일 경우 왼쪽으로 이동시킨다. 둘 중 하나라도 VOID가 아닐 경우나 t_x값이 0보다 작거나 같아지는 경우에는 이동할 수 없으므로 값을 바꾸지 않고 true를 반환한다. 여기서, 블록이 왼쪽으로만 이동하는 경우에는 테트로미노 블록의 아랫부분이 다른 테트로미노 또는 보드의 바닥에 닿지 않으므로 false는 반환하지 않는다.

- bool Board::moveBlockRight()
  블록의 좌표값을 오른쪽으로 이동시키는 역할을 하는 함수이다. 초기 t_y, t_x의 기준값은 O-블록의 왼쪽 아래 이다. 만일 현재 좌표에서 O-블록의 오른쪽 부분의 두 칸이 모두 VOID 일 경우 오른쪽으로 이동시킨다. 둘 중 하나라도 VOID가 아닐 경우나 t_x값이 width/2-1보다 크거나 같아지는 경우에는 이동할 수 없으므로 값을 바꾸지 않고 true를 반환한다. 여기서, 블록이 왼쪽으로만 이동하는 경우에는 테트로미노 블록의 아랫부분이 다른 테트로미노 또는 보드의 바닥에 닿지 않으므로 false는 반환하지 않는다.

- bool Board::moveBlockDown()
  블록의 좌표값을 아래로 이동시키는 역할을 하는 함수이다. 초기 t_y, t_x의 기준값은 O-블록의 왼쪽 아래 이다. 만일 현재 좌표가 보드의 바닥값인 경우 false를 반환하여 함수 호출을 종료한다. 아닐 경우, 현재 좌표에서 O-블록의 아래 왼쪽과 오른쪽 두 칸이 모두 VOID인지를 판별하고, t_y값이 height보다 작을 경우에만 블록의 좌표값을 이동시킨다. 둘 중 하나라도 VOID가 아닐 경우에는 블록이 더 이상 아래로 이동할 수 없으므로 새로운 블록을 호출할 수 있도록 false 값을 반환한다.

- void Board::dropBlock()
  현재 블록의 좌표값을 현재 t_x 좌표에서 가장 작은 t_y값을 가지는 t_y좌표로 이동시켜주는 함수이다. 쉽게말해 좌표값을 가능한 부분에 떨어뜨려 주는 함수이다. 좌표값 모두를 이동시키기 때문에, 일단 현재 O-블록의 4개 좌표값을 모두 지워준다. 그리고 현재 t_x좌표에서 cells[t_y][t_x] == 0 이고 cells[t_y][t_x+1] == 0 인 가장 작은 t_y좌표값을 반복문을 통해 구한다. 여기서, t_y가 board의 범위인 height-1 에 도달할 경우 반복문을 빠져나온다. 반복문을 빠져나올 경우, 두 가지 경우가 발생한다. t_y == height-1인 경우와 아닌 경우. 두 가지 경우로 나누어 if else문을 사용하여 4개의 좌표값을 찍어준다.

- void Board::eraseBlock()
  cells 배열에서 한 행의 모든 값들이 VOID가 아닌 값들로 채워져 있을 경우 지워주는 역할을 하는 함수이다. 먼저, 지워도 되는 행인지를 판별할 변수 bool erase를 선언해 준다. 그리고 각 행 별로 모두 확인하기 위해 k라는 변수를 선언하고, 그 변수의 값을 보드 높이의 최대값인 height-1(19)로 정의한다. 자세한 구현은 아래와 같다.

```
bool erase = true;

// 줄이 채워지는 경우
int k = 19;
while(k>0){
        erase = true;
        // 하나라도 VOID일 경우 지우지 않는다
        for(int j=0; j<width/2+1; j++){ // cells 0부터 9까지
                if(cells[k][j] == VOID){ erase = false; break;}
        }
        // 줄을 지우는 경우, 먼저 지우는 줄을 Void로 만들고
        if(erase){
                fulllines++; // 채워진 줄의 개수를 1 더해준다.
                for(int l=0; l< width/2+1; l++) cells[k][l] = VOID;
```

```
                Cell *tmp = cells[k]; // 지워지는 행 값을 tmp변수에 담아준다.

                int a = k;
                // 지우는 줄의 윗 줄을 하나씩 내려준다
                while(a>1){
                        cells[a] = cells[a-1];
                        a--;
                }

                cells[a] = tmp; // cell[1]에는 아까 저장해 두었던 지워진 행을 나타내는 tmp를 넣는다.

                k=20; // 하나씩 내려주니까 다시 처음부터 체크하기위해 20으로
        }
        k--;
}
```

3. BoardPane draw(Board &b)

1단계의 BoardPane draw는 BoardPane의 틀만을 그려 주었다면, 2단계 BoardPane의 draw는 이제 Board클래스의 b를 받아 이를 참조하여 내부까지 채워주어야 한다. 자세한 구현은 다음과 같다.

```
#include <ncurses.h>
#include "Pane.h"
#include "Board.h"

        BoardPane::BoardPane(int x, int y, int w, int h) : Pane(x,y,w,h){}
        void BoardPane::draw(Board &b){
                init_pair(5, COLOR_BLUE, COLOR_BLACK); // BoardPane 틀 색
                init_pair(6, COLOR_RED, COLOR_BLACK); // O-블록의 색
                init_pair(13, COLOR_BLACK, COLOR_BLACK); // VOID인 부분 지워주기 위한 색 지정
                wattron(win_, COLOR_PAIR(5));
                // win_의 1,0 좌표값부터 다이아몬드를 height -4개 찍는다(왼쪽 |)
                mvwvline(win_, 1, 0, ACS_DIAMOND, height_ -4);
                // win_의 1, width-1 좌표값부터 다이아몬드를 height -4개 찍는다.(오른쪽 |)
                mvwvline(win_, 1, width_-1, ACS_DIAMOND, height_-4);
                // win_의 height-4,0 좌표값부터 다이아몬드를 width개 찍는다.(아래 _)
                mvwhline(win_, height_-4, 0, ACS_DIAMOND, width_);

                for(int i=1; i<b.height; i++){ // 1부터 19까지
                        for(int j=0; j<b.width/2+1; j++){ // 0 부터 9까지
                                if(b.cells[i][j] == 0){ // cells의 값이 0일 경우
                                        wattron(win_, COLOR_PAIR(6));
                                        mvwhline(win_, i, 2+2*j, ACS_CKBOARD, 2);
                                        wattroff(win_, COLOR_PAIR(6));
                                }
                                else if(b.cells[i][j] == VOID){ // cells의 값이 VOID일 경우
                                        wattron(win_, COLOR_PAIR(13));
                                        mvwhline(win_, i, 2+2*j, ACS_CKBOARD, 2);
                                        wattroff(win_, COLOR_PAIR(13));
                                }
                        }
                }
                wattroff(win_, COLOR_PAIR(5));
                wrefresh(win_);
        }
```

4. InfoPane draw(Board &b)

1단계 InfoPane의 draw()에서는 Full lines와 Score에 그냥 변화하는 점수가 아닌 고정된 string값을 넣어 주었었다. 2단계 InfoPane의 draw()에서는 Board b로부터 Fulllines의 개수를 받아와 이것을 변화되는 값으로 출력한다. score는 Fullines*10으로 정한다. 기존의 draw()에서 변화되는 부분은 다음의 빨간색으로 표시되는 두 부분밖에 없다.

```
mvwprintw(win_, 2, 1, "Full Lines : ");
mvwprintw(win_, 2, width_-5, "%d", b.fulllines);
mvwprintw(win_, 3, 1, "SCORE :");
wattron(win_, COLOR_PAIR(2)); // 색깔 속성을 켠다
mvwprintw(win_, 3, width_-5, "%d", b.fulllines*10);
wattroff(win_, COLOR_PAIR(2)); // 색깔 속성을 끈다
```

5. StatPane draw(Board &b)

1단계 StatPane의 draw() 에서는 블록의 개수에 변화를 주지 않고, 고정된 string값을 넣어 주었었다. 2단계 StatPane의 draw()에서는 Board b로부터 블록의 개수를 받아와 이것을 변화되는 값으로 출력한다. 2단계에서 사용하는 블록은 O-블록 뿐이므로 O블록의 개수만을 변화시켜 준다. 변화되는 부분은 다음의 빨간색으로 표시되는 부분밖에 없다.

```
wattron(win_, COLOR_PAIR(6)); // 빨간색 켠다
mvwhline(win_, 12, 8, ACS_CKBOARD, 4);
mvwhline(win_, 12, width_/2+3, ACS_S7, 1); // -
mvwprintw(win_, 12, width_-4, "%d", b.blockcount); // 숫자
mvwhline(win_, 13, 8, ACS_CKBOARD, 4);
wattroff(win_, COLOR_PAIR(6)); // 빨간색 끈다
```

6. input.txt 받아오기

input.txt를 통해 수행한 테트리스의 결과값을 보기 위해, main함수를 새로 구성하고 Tetris 클래스에서 replay(ifstream &instream) 함수를 새로 만든다. 먼저, main함수는 다음과 같이 구현된다.

```
#include <ncurses.h>
#include <fstream>
#include "Tetris.h"
using namespace std;

int main(){

        Tetris t;
        ifstream instream;
        instream.open("input.txt"); // input.txt를 연다
        if(instream.fail()){ // input.txt에 실패할 경우
                t.play(); // 그냥 사용자에게 입력값을 받는 play()호출
        }

        t.replay(instream); // input.txt에 성공할 경우 input값을 받아 결과값을 출력

        return 0;
}
```

다음으로, 새로 만들어지는 replay(ifstream &instream) 함수는 다음과 같이 구현된다. 기존의 play와 다른 부분만 빨간색으로 표시하였다.

```
void Tetris::replay(ifstream &instream){
	char input; // input.txt의 값들은 char형태이므로 int형이었던 input변수를 char형으로 바꾼다.
	bool gameOver = false;
	bool newBlock = false;
	updateScreen();
	input = getch();


	while(1){ // 새로운 블록을 생성하는 반복문
		b.eraseBlock(); // 일단 꽉찬블록 다 지워주고
		if(gameOver){ // 'q'값이 input으로 들어왔을 경우
			input = getch();
			break;
		}
		gameOver = !b.insertBlock();
		updateScreen();
		if(gameOver){
			input = getch();
			break;
		}
		while(!newBlock){
			instream >> input; // instream으로부터 input값을 받아준다.
			switch(input){
				case 'l': // 'l'을 받는 경우
					newBlock = !b.moveBlockLeft();
					break;
				case 'r': // 'r'을 받는 경우
					newBlock = !b.moveBlockRight();
					break;
				case 'g': // 'g'을 받는 경우
					newBlock = !b.moveBlockDown();
					break;
				case 'd': // 'd'을 받는 경우
					b.dropBlock();
					newBlock = true;
					break;
				case 'q': // 'q'을 받는 경우
					newBlock = true;
					gameOver = true;
					break;
			}
			updateScreen();
		}
		newBlock = false;
	}
}
```

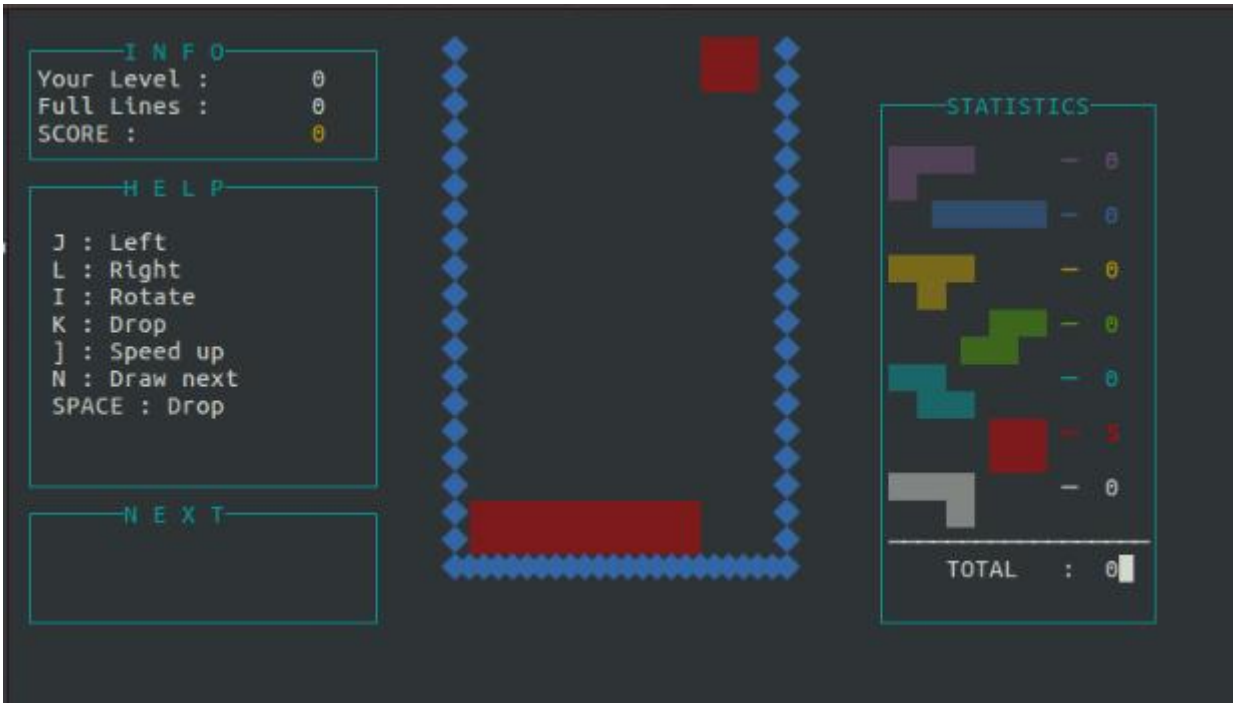
7. MakeFile 만들기

```
CC = g++
TARGET = tetris
SOURCES = Tetris.cpp \
          Pane.cpp \
          BoardPane.cpp \
          InfoPane.cpp \
          HelpPane.cpp \
          StatPane.cpp \
          NextPane.cpp \
          Board.cpp \
          main.cpp
LDFLAGS = -lncurses
all:
$(CC) -o $(TARGET) $(SOURCES) $(LDFLAGS)
clean:
rm -rf *.o $(TARGET)
```

위와 같은 형태로 Makefile을 만들어준 후, 터미널 창에서 make 명령어를 통해 컴파일을 실시한다.

8. 결과물

- replay(ifstream &instream) 수행 시



– play() 수행 시 : Full Lines와 Score 변화