# 테트리스 코드 리뷰

12171742 최진우

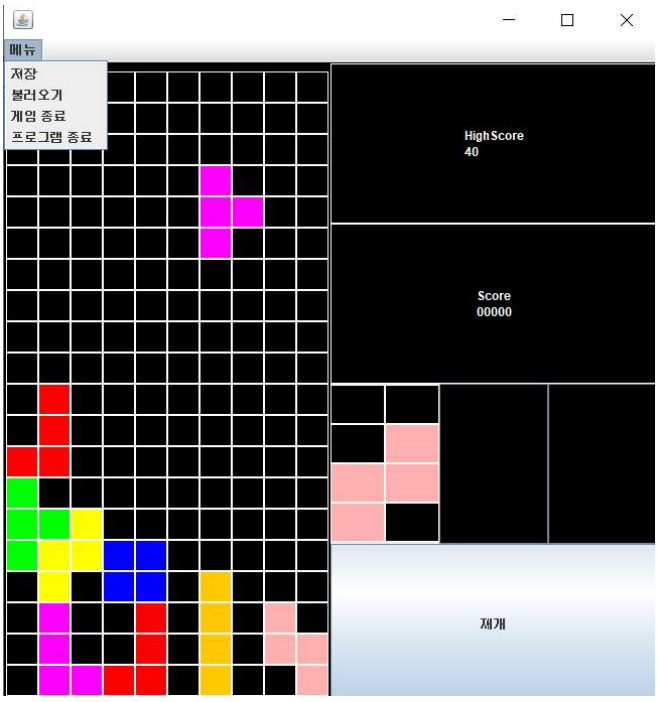
목차



## 기본 구조

이 프로그램은 두개의 소스코드로 이루어져있다. Main.java와 UI.java이다. Main.java는 주된 동작을 담당하는 코드이고 Ui.java는 swing ui를 담당하는 코드이다. Ui.java는 windowBuilder를 활용하여 자동으로 제작된 코드에 좌측의 캡쳐본과 같이 for문을 활용하여 game이라는 이름을 가진 JPanel에 JButton들을 10*20개 추가해 주었다. 또한 getThings라는 함수가 사용되어 string으로 입력받은 요소명에 해당하는 요소를 return 해준다. Main.java는 이러한 Ui에서 직접적인 로직을 구현하는 코드이다. Main클래스와 block 클래스가 있는데 먼저 Main클래스를 확인해보겠다. Main 클래스에서는 Ui를 불러온 후 키보드이벤트, 마우스클릭이벤트 등의 리스너들을 프레임 및 버튼들에 붙여준다. 이후 block클래스를 생성 후 실행하고 종료된다. block클래스가 실질적인 로직이 들어가있는 클래스라고 할 수 있는데 여기서 블록의 이동, 회전 등의 여러 이벤트들을 제어한다. 블록의 이동은 키보드입력을 통해 제어가능하며 키보드 좌우 방향키로 좌우 이동, 키보드 아래 방향키를 누른만큼 아래로 빠르게 이동, 키보드 위쪽 방향키로 블록회전을 시현할 수 있다. 좌측에 보이는 게임화면을 통해 내용을 참고하면 좋을 것이다. 현재 내려오는 분홍색 블록을 앞서 말한대로 키보드로 조작하여 흔히 알려진 테트리스 게임을 진행할 수 있다. 우측 상단의 하이스코어는 파일로 저장된 최고점수 파일을 게임실행 시 불러와서 시현해준다. 그 아래 스코어는 현재 플레이어의 점수를 나타내주는데 한 줄을 없앨때마다 10점씩 쌓이게 된다. 점수패널 밑의 창은 다음 번에 나타날 블록을 미리 보여주는 곳 이다. 그 아래의 일시정지/재개 버튼은 써있는 그대로 게임을 일시정지하고 이후 재개할 수 있는 버튼이다. 상단의 메뉴버튼은 4개의 버튼을 포함하고 있다. 저장 버튼은 현재 게임상태를 저장해준다. 불러오기버튼은 이전에 저장했던 게임상태를 불러와서 그 시점부터 시작한다. 게임종료버튼은 게임만 종료되고 프로그램은 종료되지 않는다. 프로그램 종료버튼은 프로그램을 종료시킨다. 물론 우측 상단의 X 표시를 누르면서 종료할 수도 있다.

```
game.setLayout(new GridLayout(20, 10, 0, 0));
for (int i = 0; i < 20; i++) {
    for (int j = 0; j < 10; j++) {
        block[i][j] = new JButton();
        JButton tmp = block[i][j];
        tmp.setBorder(gameLine);
        game.add(tmp);
        tmp.setName((i+1) + "_" + (j+1));
        tmp.setBackground(Color.black);
    }
}
```



## 함수 설명

Main.java 함수를 확인해보면 run이 메인으로 돌아가고 있고 많은 플래그들이 상태를 나타낸다. 이를 다음과 같이 표로 정리할 수 있다. 이러한 플래그들은 해당 기능을 수행한 뒤 다시 false로 돌아가 각자 연결된 리스너에서의 입력을

| 이름 | 기능 |
|---|---|
| movingRight | 우측키보드버튼을 누를 시 해당리스너가 true로 변경시켜줌 |
| movingLeft | 좌측키보드버튼을 누를 시 해당리스너가 true로 변경시켜줌 |
| rotate | 상단키보드버튼을 누를 시 해당리스너가 true로 변경시켜줌 |
| down | 하단키보드버튼을 누를 시 해당리스너가 true로 변경시켜줌 |
| save | 저장버튼 클릭 시 해당리스너가 true로 변경시켜줌 |
| load | 불러오기버튼 클릭 시 해당리스너가 true로 변경시켜줌 |
| over | 블록들이 쌓여서 더 이상 게임을 진행할 수 없을 때 해당리스너가 true로 변경시켜줌 |
| pause | 일시정지버튼 클릭 시 해당리스너가 true로 변경시켜줌 |
| nowOver | 게임 종료 버튼 클릭 시 해당리스너가 true로 변경시켜줌 |

기다리게 된다. 각각의 플래그들은 run 내부의 while(true)문이 돌아가면서 계속하여 확인한다. 이어서 그 외의 변수들에 대해 확인해 보면 다음과 같다. 정리하자면 매 7번마다 blockColor와 order배열을 섞은 후 여기서 count가 0부터

| 이름 | 기능 |
|---|---|
| blockColor | 7가지의 Color 객체를 저장한 후 순서를 섞어 블록에 색상을 입힌다. |
| order | 7가지 블록의 모양을 1에서 7로 나타내어 저장하였다. 이 순서를 섞어서 블록의 모양을 정한다. |
| shape | 블록이 회전할 때 최대 4가지의 모양이 나오는데 이것을 1에서 4로 나타내어 저장하였다. 앞서 언급한 rotate가 true로 될때마다 1부터 4까지 하나씩 순환 후 1로 돌아오는 것을 반복한다. |
| curPos | 블록의 현재 좌표를 (y,x) 형식으로 리스트에 저장한 후 이를 블록이 이동시마다 업데이트 한다. |
| count | 순번을 나타내주는 변수로서 앞서 섞은 색깔과 모양들을 0번인덱스부터 6번인덱스가지 출력하기 위해 블록 하나를 만들 때마다 카운트를 하나씩 올리면서 접근한다. 끝까지 올라간 후에는 다시 0으로 바뀐다. |
| myScore | 앞서 언급한 점수를 파악하기 위한 변수이다. 현재 나의 점수를 지속하여 업데이트 한다. |

6까지 돌게 된다. 각각의 리스트에서 count번 인덱스요소를 뽑아내어 이를 블록의 색상과 모양으로 사용한다. 이렇게 되면 랜덤한 모양과 색상의 7세트의 블록이 매 세트마다 랜덤하게 반복되어 나온다. 출발 지점 또한 x좌표를 random 함수를 통해 지정하여 매 블록마다 랜덤하게 지정한다. 이렇게 블록을 랜덤하게 생성한 후 기본적으로 모든 플래그들이 false이면 while문을 통해 바닥이 나올 때 까지 아래로 내려가게된다. 이때 이 내려가는 과정은 블록을 하나씩 색칠

```
public void dropBlock() {
    Color col = blockColor.get(count);
    if ((order.get(count) == 3 && shape == 2) || (order.get(count) == 1
            (order.get(count) == 2 && shape == 4) || (order.get(count)
        for (int i = 0; i < 4; i++) {
            JButton tmp = Ui.getThing(curPos[i][0] + "_" + curPos[i][1]
            tmp.setBackground(Color.black);
            tmp = Ui.getThing((curPos[i][0]+1) + "_" + curPos[i][1]);
            tmp.setBackground(col);
        }
    }
    else {
        for (int i = 3; i > -1; i--) {
            JButton tmp = Ui.getThing(curPos[i][0] + "_" + curPos[i][1]
            tmp.setBackground(Color.black);
            tmp = Ui.getThing((curPos[i][0]+1) + "_" + curPos[i][1]);
            tmp.setBackground(col);
        }
    }
    for (int i = 0; i < 4; i++) {
        curPos[i][0] += 1;
    }
```

하고 지우는 과정인데 좌측의 코드 일부분의 캡처본을 참고해보겠다. 블럭 각각의 형태에 아래로 내려지는 작업을 수행할 때 curPos의 3번째좌표부터 색을 바꾸기 시작해서 0번째좌표를 마지막으로 바꿀지, 아니면 그 반대일지에 따라 정상작동의 여부가 갈린다. 이에 분기문을 활용하여 해당 경우를 여과한다. 현재좌표의 버튼 색상을 검은색으로 변경 ->현재좌표의 y축+1에 해당하는 버튼 색상을 현재 버튼의 색상으로 변경->현재좌표를 색상 변경한 좌표로 업데이트 와 같은 방식으로

이동이 이루어진다. 좌측, 우측, 회전 모두 현재좌표에서 원하는 좌표로 이동 후 변경하는 부분에서만 맞는 자리로 이동하는 좌표계산이 다를뿐 기본적으로는 이와 같은 방식을 사용하고 있다. 좌,우측 이동, 블록회전은 아래의 해당 함수의 일부분을 통해 볼 수 있다. 첫번째 캡쳐본은 좌우이동에 관한 함수이다. 하향이동과 마찬가지로 특수한 경우를 예외로 처리하는 것을 볼 수 있다. 두번째 캡쳐본은 블록회전에 관한 함수의 일부이다. 다른 두 함수와는 달리 분기가 많은데 블록의 개수가 7개이고 회전하는 모양까지 고려하면 19가지가 되기 때문이다. 또한 하나의 블록에 대해서도 최대 4가지의 모양을 고려해야 하니 분기가 많아지게 된다. 이렇게 여러 함수들이 실행되며 while문을 통해 블록이 계속해서 옮겨지는데 어느 시점에서는 멈추고 다음 블록을 준비해야 될때가 올 것이다. 이를 판별해주는 것이 isFloor함수로 블록의 다음색깔이 배경색이 아닌 유색인지, 다음번 이동위치가 10*20의 범위를 벗어났는지등을 고려하여 멈춰야하는지 말아야하는지를 알려준다. 이러한 정보를 바탕으로 block의 run함수는 제때 다음 블록을 준비 할 수 있게 된다. 다른 기능으로 일시정지와 블록미리보기가 있는데 생각보다 간단하게 구현되어있다. 일시정지는 버튼을 누를시 pause 플래그가 true로 변한다. 이때 run은 매 루프마다 while(true) { if (pause == false) break;} 와 같은 구문을 지나게 되는데 pause플래그가 true로 변한 시점에서 run은 해당 while문에 갇히게 된다. 버튼을 한 번 더 누르면 pause는 true 가 되면서 while문을 빠져나오게 되고 run은 정상적으로 다시 루프를 돌게된다. 블록미리보기는 앞서 언급한 count를 통한 색상배열과 모양배열로의 접근에서 count가 아닌 count+1로 배열에 접근함으로서 구현한다. 이렇게 가져온 데이터를 별도로 마련된 공간에서 시현하는 것이다. 이제 게임 종료부분이다. 게임 종료는 강제종료와 자동종료가 있다. 자동종료는 블록이 맨 위까지 차올라서 더 이상 움직일 수 없는 상황일 때 발생하고 강제종료는 파일의 게임종료버튼을 눌렀을 때 발생한다. 자동종료는 블록의 상태를 체크하는 함수가 over플래그를 true로 만들었을 때 highscore와 현재스코어를 비교후 현재스코어가 더 크다면 하이스코어 업데이트를 한다. 이후 팝업메시지에 사용자의 점수를 출력해주고 게임이 종료된다. 강제종료는 게임 종료버튼을 눌렀을 때 nowOver 플래그가 true가 되고 이 때 앞서 언급한 하이스코어와의 비교 및 업데이트를 실행하게 된다. 이 후 팝업메시지로 사용자의 점수를 출력하고 게임은 종료된다. 이러한 사용자 점수는 checkBreak 함수를 통해 게임중 집계되는데 해당 함수는 모두 유색이 된 줄이 있는지 체크 하고 있으면 해당 줄의 색상을 검은색으로 변경 후 위쪽의 색상들을 아래로 끌어내려준다. 이때 점수가 10점씩 올라가게 되는데 setText로 즉시 스코어화면에 시현된다. 마지막으로 저장기능이다. 메뉴의 저장버튼을 클릭 시 save플래그가 true로 변하여 save함수를 호출하게 되는데 이 함수에서는 게임의 10*20배열의 색상정보와 현재 좌표들이 color//color//...//color//(x,y)//...//(x,y) 형식의 string으로 텍스트파일에 저장이 된다. 이 데이터는 불러오기 버튼이 클릭되었을 때 load플래그가 true로 변하여 load함수를 통해 가공된다. 해당 string은 //을 기준으로 split되어 색상과 현

```
if ( (order.get(count) == 2 && shape == 3)) {
    for (int i = 0; i < 4; i++) {
        JButton tmp = Ui.getThing(curPos[i][0] + "_" + curPos[i][1]);
        tmp.setBackground(Color.black);
        tmp = Ui.getThing(curPos[i][0] + "_" + (curPos[i][1]+1));
        tmp.setBackground(col);
    }
}
else if (start == 3 || (order.get(count) == 1 && shape == 2) || (order.get(count) == 2 && sha
        start == 2) {
    for (int i = 3; i > -1; i--) {
        JButton tmp = Ui.getThing(curPos[i][0] + "_" + curPos[i][1]);
        tmp.setBackground(Color.black);
        tmp = Ui.getThing(curPos[i][0] + "_" + (curPos[i][1]+1));
        tmp.setBackground(col);
    }
}
else {
    for (int i = 0; i < 4; i++) {
        JButton tmp = Ui.getThing(curPos[i][0] + "_" + curPos[i][1]);
        tmp.setBackground(Color.black);
```

```
if (order.get(count) == 0) {
    if (shape == 1) {
        if (curPos[1][1]-1 < 1) return;
        if (Ui.getThing((curPos[0][0]+1) + "_" + (curPos[0][1]-1)).getBackground() != Color.black
        JButton tmp = Ui.getThing(curPos[0][0] + "_" + curPos[0][1]);
        tmp.setBackground(Color.black);
        tmp = Ui.getThing((curPos[0][0]+1) + "_" + (curPos[0][1]-1));
        tmp.setBackground(col);
        curPos[0][0] += 1;
        curPos[0][1] -= 1;
        int tmp1 = curPos[2][0];
        int tmp2 = curPos[2][1];
        curPos[2][0] = curPos[3][0];
        curPos[2][1] = curPos[3][1];
        curPos[3][0] = tmp1;
        curPos[3][1] = tmp2;
        shape = 2;
    }
    else if (shape == 2) {
        if (curPos[1][0]-1 < 1) return;
        if (Ui.getThing((curPos[1][0]-1) + "_" + (curPos[1][1])).getBackground() != Color.black)
        JButton tmp = Ui.getThing(curPos[2][0] + "_" + curPos[2][1]);
        tmp.setBackground(Color.black);
        tmp = Ui.getThing((curPos[2][0]-1) + "_" + (curPos[2][1]-1));
        tmp.setBackground(col);
```

재좌표가 다시 배열에 할당된다. 해당경로는 c:/homeWork/tetData.txt이다. 앞서 highscore의 저장경로는 c:/homeWork/tetScore.txt로 두 데이터는 서로 달리 저장된다.