JAVA TETRIS PROJECT

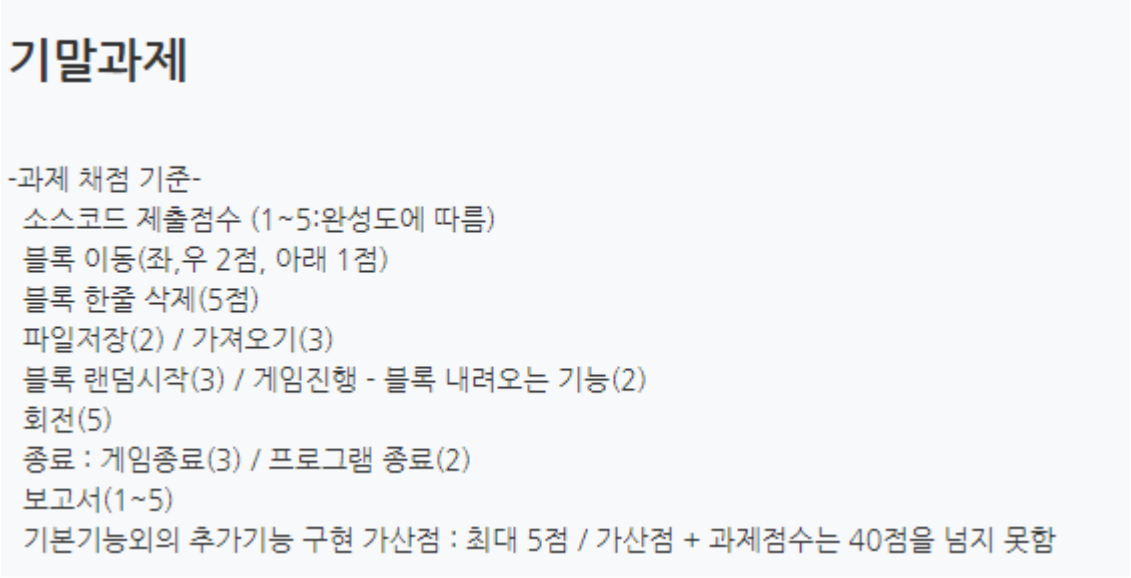
과목명 자바기반응용프로그래밍

학과 컴퓨터공학과
학번 12162149
이름 정원철
제출일자 2020.12.19

Table of content

주제 분석(Assignment topics analysis)

- **구성 계획설명 및 과제 목표**



이번에 진행하게 된 과제는 중간고사 이후 범위에서 계속 다루고 배웠던 GUI를 활용한 테트리스 게임 구성이다. 주어진 평가지표에도 나와 있듯이 테트리스 게임이 가져야할 Standard한 기능을 요구하고 있다. 천천히 하나씩 살펴보자.

첫 번째는 블록의 이동, 사실상 테트리스 게임에 있어서 가장중요한 기능 중 하나다. 게임을 벗어 나지 않는 범위내에서 블록의 하, 좌, 우 이동을 컨트롤 할 수 있어야한다. 이 부분에 있어서는 예상컨데 키보드의 방향키 입력을 받아 이벤트처리로 구성을 할 것 같다. 여기서 중요한 포인트는 아래로 이동하는 것인데 기본적으로 테트리스는 별도의 조작이 없으면 자동으로 블록이 일정 템포에 맞춰 아래로 떨어지게 되어있다. 하지만 키보드 조작 (이벤트 처리)을 통해 아래로의 이동을 컨트롤한다면 기본으로 설정된 낙하속도보다 빠르게 떨어지도록 처리해야 한다. 이부분이 키 포인트라고 말할 수 있을 것 같다.

두 번째는 누적된 블록의 삭제이다. 테트리스는 바닥에서 한 행이 빈틈없이 채워지면 그 줄은 사라지고 그 위에 있던 줄이 차례로 삭제된 줄의 수만큼 내려온다. 보통 이 부분은 게임창에서 설정한 사이즈를 기반으로 바닥에서부터 (Y=0 인 파트) 차례로 한 줄씩 체크해가는 이차원 행렬의 비교연산을 통해 처리가 될 것 같다. 여기서 중요한 키포인트는 한 줄이든, 두 줄이든 없어져야 되는 상황이면 그 위에 누적 되어있던 블록들은 그 형태 그대로 바닥으로 정렬되어야 한다는 것이다.

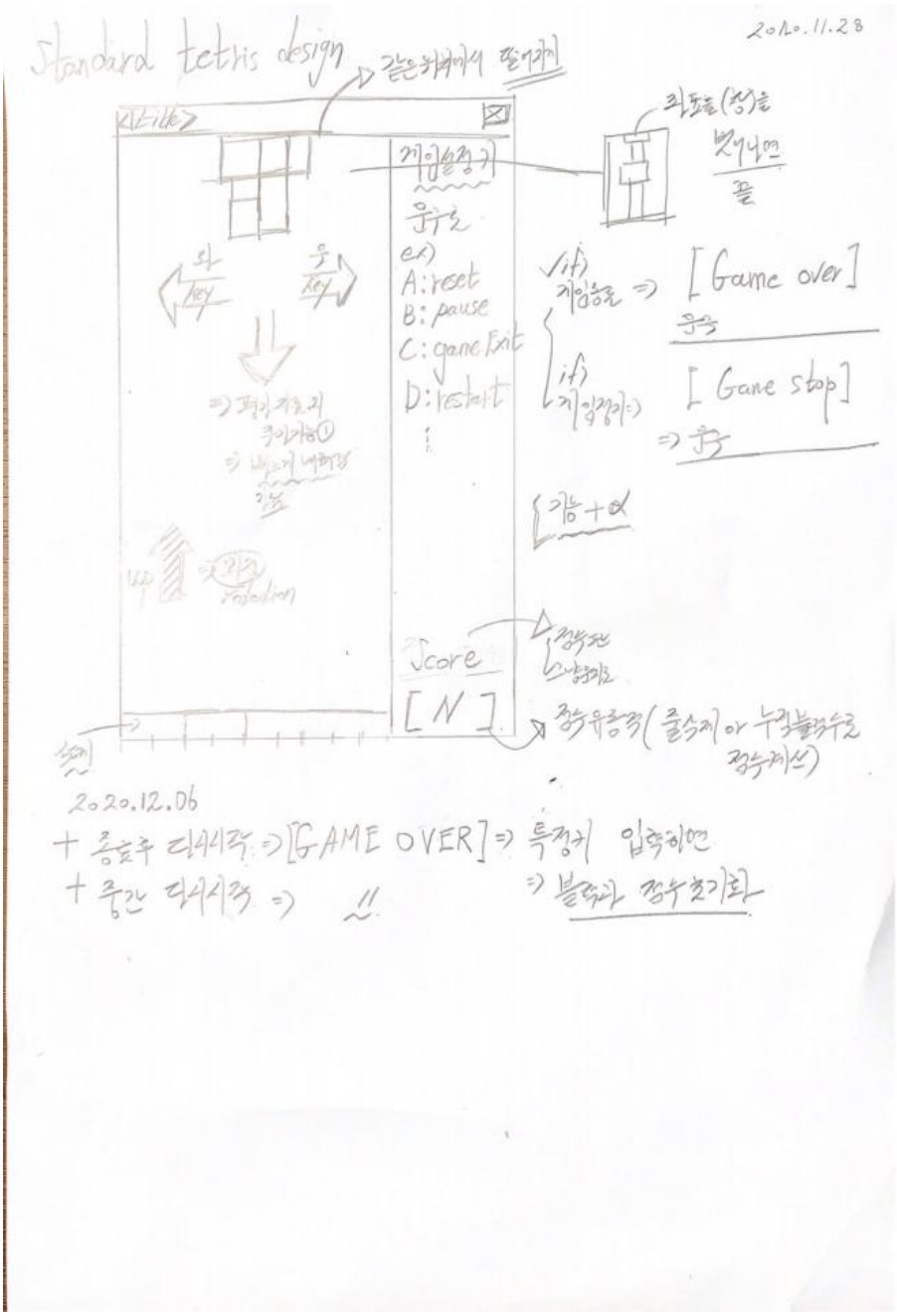
세 번째는 파일저장과 가져오기인데, 우리가 배웠던 파일 입출력을 활용한 게임저장 장치를 만드는 것 같다. 우리가 게임을 진행하고 있을 때 종료를 하거나, 특정 외부 저장처리를 하게 된다면 딱 그 시점까지 게임을 저장하고, 다시 불러올 때 마지막으로 진행했던 구간을 다시 불러오는 기능을 수행해야한다. 보통 테트리스는 좌표를 통한 작업을 하기에 배열정보를 활용한 구성을 진행해야 할 것 같다. 게임창의 사이즈를 [너비][높이]식으로 이차원 배열식으로 구성하기에 반복문을 통한 해당배열의 모든 좌표를 탐색해 파일에 쓰고 (블록이 있는 부분이면 true(=1), 없는 부분이면 true(=0))다시 불러올 때는 reader를 통해 다시 저장한 파일의 값들을 해당 배열에 삽입해 저장된 부분을 구현하게 될 것 같다.

마지막은 회전, 블록 랜덤생성, 블록 하강이다. 일단 나는 좌표에 색을 계속 칠하는 과정을 반복해 구성할 예정이기에 블록이 일정시간마다 한 칸씩 내려올 수 있게 좌표 값을 하나씩 낮추며 다시 칠하는 과정을 반복해 구성할 것 같다. 회전은 배열을 통해 저장해 놓은 블록 구성을 말그대로 역순으로 바꿔 처리해야 한다. 여기서 배열 수치만 바꾸게 되면 회전을 해서는 나올 수 없는 예외적인 경우가 등장하기에 다른 메서드나 코드를 통해 처리해야 할 것이다. 실제 테트리스 게임에서도 블록은 랜덤으로 우리가 다음에 무슨 모양의 블록이 나올지 예측할 수 없는 구조다. 그래서 사실 테트리스에서 사용하는 7개의 블록을 7개의 객체로 저장해 랜덤처리를 하게 된다면 충분히 구현이 가능할 것으로 생각이 된다.

이번 테트리스 과제의 초점은 "**원초적 기능**"에 두려고 한다. 사실 위 기능만 구현할 수 있게 된다면 추가적인 기능은 사실상 기본적으로 따라올 수밖에 없기에 standard한 기능 구현에 초점을 맞추겠다.

주제 분석(Assignment topics analysis)

- 사전 GUI Design



구상해본 기능적 GUI 디자인은 위와 같다.

1. 게임을 진행하는 영역과 스코어와 다른 설명문 등이 들어갈 영역을 구분
2. 누적되는 블록의 수마다 혹은 삭제되는 행의 수 마다 스코어 증가
3. 특정 상황 정지, 게임 종료 시 등장할 문구 설정
4. 게임진행 외부영역 추가 문구설정

Tetris Code Review (3)

이번 **프로젝트의 파일 총 3개**로 구성 되어있다. 전체적인 GUI 외형 틀을 잡는 tWindows.java, 전반적인 게임기능을 수행하는 Shipping.java, 문구나 이벤트처리, 블록모양 설정 등의 기능을 포함하고 있는 tBoard.java, 이 3가지의 파일로 구성되어 있는데 우선 **tWindows.java**부터 같이 살펴보자. 이미 소스파일에 주석처리가 대부분 되어있기에 **중요하다고 생각하는 파트를 위주**로 소개하겠다.

- **tWindow.java**

```
package wonchel.jeong.assignment;

import javax.swing.JFrame;

public class tWindows {
    private JFrame twindow;
    private tBoard board;

    public tWindows() {
        twindow = new JFrame("Woncheol's Tetris"); // jFrame을 통해 상단 title바에 적당한 이름을 설정한다
        twindow.setSize(480, 640); // 게임창 사이즈를 설정한다
        twindow.setResizable(false); // 창 설정후 마우스 드래그를 이용한 사이즈 크기 설정 불가
        twindow.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE); // 실제 창의 종료 버튼인 x를 누르면 종료되는 기능
        twindow.setVisible(true); // 유저의 눈에 보이게 설정 화면 지정
        board = new tBoard(); // 보드 객체를 할당
        board.BlockFileCall(); //이전까지 저장되었던 테트리스게임의 블럭기록을 가져오는 메소드이다, 실행되는 순간 저장된 기록이 연속해서 실행된다
        board.TetrisCall(); // 이전까지 저장되었던 테트리스게임의 점수기록을 가져오는 메소드이다, 실행되는 순간 저장된 기록이 연속해서 실행된다
        twindow.addKeyListener(board); // tBoard에서 처리한 이벤트처리를 추가한다
        twindow.add(board); // 마찬가지로 보드의 전반적인 내용을 추가한다

    }

    public static void main(String[] args) {

        new tWindows(); // 생성자를 통한 게임실행

    }

}
```

전반적으로 이 페이지에서 다루는 것이 타 파일에서 구현한 정보를 추가하는 역할을 하기 때문에 복잡한 코드나 구성은 보이지 않는다. 대부분 이미지의 주석처리로 충분히 이해할 수 있게 구성했으며, 구성하면서 처음 써본 메서드는 `setResizable()`이다. 처음 게임창의 크기를 설정할 때는 플레이나 미관상 보기 좋게 최적화된 설정을 목적으로 하는데 보통 프레임은 마우스로 드래그를 하게 되면 유동적으로 조절되어 디자인한 구성이 무너 질 수가 있다. 하지만 위 함수를 통해 Boolean 매개 변수를 통해 사이즈 설정의 가능여부를 설정할 수 있게 구성해 놓았다. `board.TetrisCall()`의 코드 구현은 tBoard.java에서 살펴볼 수 있겠지만 이 메서드를 tWindow.java파일에 추가함으로써 게임시작과 동시에 저장된 게임의 점수를 불러올 수 있다. `board.BlockFileCall()` 도 사실 위 코드와 마찬가지로 게임내용을 불러오는 기능을 수행한다. 위 메소드는 점수를 불러오는 기능을 했다면 **BlockFileCall()**은 누적된 블록의 기록까지도 가져와 게임을 실행할 수 있다.

Tetris Code Review

- **tBoard.java(1)**

```
package wonchel.jeong.assignment;

import java.awt.Color;

public class tBoard extends JPanel implements KeyListener {

    // 게임 상태를 나타내는 나타내는 상수들이다. 게임의 상태를 특정 숫자로 지정함으로써 간단한 상황 설정이 가능하다.
    public static int STATE_GAME_END = 0;
    public static int STATE_GAME_STOP = 1;
    public static int STATE_GAME_PLAYING = 2;
    private int state = STATE_GAME_PLAYING;


    // 게임을 진행하는 공간의 사이즈를 설정해줄 상수들이다
    public static final int SET_tBOARD_HEI = 20; // 게임 내 창 높이를 상수로 선언
    public static final int SET_tBOARD_WID = 10; // 게임 내 창 너비를 상수로 선언
    public static final int SET_tSPACE_SIZE = 30; // 가상의 칸을 30으로 설정후 상수로 선언


    // 기본적으로 블록이 자동으로 낙하하는 시간을 지정해둔 상수다
    private static int SET_DELAYTIME = 60 / 1000; // 딜레이 타임 설정


    // 칸이 내려올때 마다 계속 색을 다시 색칠하는 방식이기에 게임보드의 넓이에 맞게 설정한다
    private Color[][] board = new Color[SET_tBOARD_HEI][SET_tBOARD_WID];


    private Random random; // 게임내에서 테트리스 도형이 랜덤으로 나오게 처리하기 위해 랜덤 상수선언
    private int x = 3, y = 0; // 게임은 x,y축 좌표기반으로 진행이되기에, 초기 블럭위치 좌표를 설정해준다.

    private int CHANGE_X; // 키보드 이벤트를 통한 좌우 이동에 (x좌표 이동) 쓰일 변수다
    private boolean BUMPER = false; // BUMPER는 상하와 관련된 충돌과 관련된 boolean이다. 블럭의 누적 여부를 좌우한다.

    private Timer dropping; //설정 시간 주기에 맞춰 같은 반복을 하게되는 Timer 클래스의 변수를 선언한다. 일종의 쓰레드의 역할을 Timer가 해준다.

    private Color[] colors = { Color.green, Color.cyan, Color.pink, Color.orange, Color.BLUE, Color.magenta,
            Color.YELLOW }; //게임내에서 사용하는 7가지의 블록의 색을 지정하는 컬러 배열이다. tShape[i] (0<i<7) 차례로 색이 설정된다.

    private Shaping[] tShape = new Shaping[7]; // 테트리스 게임에 사용될 도형들을 담아두기위한 사이즈의 배열을 할당받는다
    private Shaping curShape;                  // 게임중 당시 바로 사용되는 블록을 나타내주는 역할
    public static int score = 0;               // 게임 스코어를 받아줄 정수형 변수다
```

자세한 구현 내용은 소스파일의 주석처리를 통해 확인할 수 있다. 각 소스 게임 구현에 있어서 중요한 부분을 위주로 설명하려 한다. 위 소스는 각 메소드나 코드처리들을 위해 선언한 재료들이다. 코드를 확인해보면 `SET_DELAYTIME` 이 눈에 띌 수가 있는데, 이는 쓰레드를 대신해 이전에 개인공부 시간 때 학습한 적이 있는 Timer 클래스를 사용해 낙하처리를 진행하려고 선언한 변수다. 이 외에는 게임상황에 따른 [1]정지, [2]진행, [0]종료 상황을 숫자로 대입해 적용시키는 부분을 메인 파트라고 볼 수 있을 것 같다.

이제 본격적으로 각 소스파일의 중요한 부분을 살펴보자.

Tetris Code Review

- **tBoard.java(2)**

-블록 구성 파트

```
// 테트리스 블록의 모양과 컬레를 설정하는 구간이다
tShape[0] = new Shaping(new int[][] { { 1, 1, 0 }, { 0, 1, 1 }, // Z 모양;
}, this, colors[5]);

tShape[1] = new Shaping(new int[][] { { 1, 1, 1 }, { 0, 0, 1 }, // 기역자 모양 ;
}, this, colors[3]);

tShape[2] = new Shaping(new int[][] { { 1, 1, 1, 1 } // 1자 모양;
}, this, colors[0]);

tShape[3] = new Shaping(new int[][] { { 1, 1 }, { 1, 1 }, // 네모 모양
}, this, colors[6]);

tShape[4] = new Shaping(new int[][] { { 1, 1, 1 }, { 0, 1, 0 }, // T자 모양;
}, this, colors[1]);

tShape[5] = new Shaping(new int[][] { { 0, 1, 1 }, { 1, 1, 0 }, // S 모양;
}, this, colors[4]);

tShape[6] = new Shaping(new int[][] { { 1, 1, 1 }, { 1, 0, 0 }, // 리버스 기역자 모양 ;
}, this, colors[2]);

curShape = tShape[0]; // 게임 시작시 첫번째로 등장하는 블럭은 tShape[0]->Z 모양 블럭으로 설정한다. 이후 블럭은 랜덤으로 설정.
```

테트리스 게임에서 사용되는 블록의 종류는 총 7가지이다. 좌표를 통한 게임을 진행이 이뤄지기 때문에 2차원 배열 구성을 통해 각 블록의 모양을 설정하고, 이전에 3개의 매개변수를 shaping 생성자를 통해 받아 설정한다.

```
public Shaping(int[][] coor, tBoard board, Color color) {
    this.coor = coor;
    this.board = board;
    this.color = color;

}
```

처음 생성되는 블록의 모양은 tShape[0]로 지정해 두었다. Z 블록으로 고정 설정을 해놓고 이후 블록부터 랜덤으로 생성되어 배치되는 구조다. 블록의 랜덤배정 파트는 이후 코드에서 확인해보자.

Tetris Code Review

- **tBoard.java(3)**

-입, 출력파트

```
public void BlockFileSave() { // 블럭의 누적기록을 저장하는 메서드다.

    String colorCheck = null;
    try {
        FileWriter out = new FileWriter("D:\\Tetris1.txt"); // 파일 라이터를 통해 지정된 경로에 파일이 생성되도록한다.
        for (int i = 0; i < board.length; i++) {            //
            for (int j = 0; j < board[0].length; j++) {
                // 1, Color.green, 2, Color.cyan, 3, Color.pink,
                // 4, Color.orange,5, Color.BLUE, 6, Color.magenta, 7, Color.YELLOW

                if (board[i][j] == Color.green) {           // 보드 배열은 컬러배열이기에 파일에 쓰기위해선 별도의 변환이 필요하다
                    colorCheck = "1";                       // 배열에서 추출된 컬러에 적당한 숫자별명을 매겨 문자로 파일에 write를 해준다.
                } else if (board[i][j] == Color.cyan) {     // 빈칸의 경우에는 별도로 0이라는 문자로 처리해보았다.
                    colorCheck = "2";
                } else if (board[i][j] == Color.pink) {
                    colorCheck = "3";
                } else if (board[i][j] == Color.orange) {
                    colorCheck = "4";
                } else if (board[i][j] == Color.BLUE) {
                    colorCheck = "5";
                } else if (board[i][j] == Color.magenta) {
                    colorCheck = "6";
                } else if (board[i][j] == Color.YELLOW) {
                    colorCheck = "7";
                } else {
                    colorCheck = "0";
                }
                out.write(colorCheck);
            }
        }
        out.close();
    } catch (IOException E) {
    }
}
```

```
public void BlockFileCall() { // 블럭누적기록을 불러오는 메소드다.

    Color[] colorzip= new Color[200];       //컬러들을 받을 별도의 컬러 배열을 할당해준다
    int color;                              //파일을 정수로 읽어들일 것 이기때문에 별도의 정수 변수 설정
    int conut=0;                            //배열에 차례로 반복문을 통해 값을 삽입할것 이기에 카운트설정
    Color color2 = null;
    try {
        File log_file = new File("D:\\Tetris1.txt"); // 해당경로에 기록된 파일을 할당
        FileReader in = new FileReader(log_file);
        int count=0;
        while (true) {
            color = in.read();                      // 파일을 하나씩 읽어 color에 삽입한다.
            if (color == -1)                        // 파일의 끝이 보이면 반복문을 끊어 읽기를 마무리해준다.
                break;

            if (color == 48) {                      // 빈칸을 위에서 0이라고 저장했기에 컬러기준 null 처리를 해준다.
                color2 = null;
            } else if (color == 49) {               // 문자를 정수로 읽어오면서 아스키코드로 변환이되는데 총 7가지의
                color2 = Color.green;               // 색이 존재하기때문에 각 넘버마다 색을 다시 입혀준다
            } else if (color == 50) {
                color2 = Color.cyan;
            } else if (color == 51) {
                color2 = Color.pink;
            } else if (color == 52) {
                color2 = Color.orange;
            } else if (color == 53) {
                color2 = Color.BLUE;
            } else if (color == 54) {
                color2 = Color.magenta;
            } else if (color == 55) {
                color2 = Color.YELLOW;
            }

            colorzip[count]=color2;                 // 선언해놓은 배열에 받은 컬러들을 모아준다

            count++;
        }
        int k=0;
        for (int i = 0; i < board.length; i++) {
            for (int j = 0; j < board[0].length; j++) {
                board[i][j]=colorzip[k++];          // 마지막으로 보드 배열에 컬러를 차례대로 삽입해 누적되었던 블록의
                                                    // 기록들을 찾아온다
            }
        }

        in.close();
    } catch (IOException E) {
    }
}
```

사실 이 두 파트, `BlockFileCall(),BlockFileSave()`가 가장 신경을 많이 쓴 구간이다. 처음 코드를 구현할 때 입출력에 관한 생각을 배제하고 코드를 짰었기에 일반적인 좌표개념 뿐만 아니라 색까지 입히는 작업으로, 좌표를 구성하는 Board 배열은 Color 배열로 선언하게 되었다. 이부분에서 누적된 블록의 위치를 찾는 건 문제가 되지 않지만 writer 하는 과정에서 파일에 color 자체를 여과없이 저장할 수 없기에, 사용된 컬러를 모아 각 숫자가 갖는 일종의 별명을 지어, 정수 문자열로 파일에 write를 진행했다.

이후 파일을 read하는 과정에서는 다시 컬러로 불러와야 하기에 이전에 지정해 두었던 별명을 토대로 변환해, 다른 컬러 배열을 설정하고 내용을 담아 넣은 후 다시 보드배열의 사이즈에 맞게 재 삽입해 파일을 읽어올 수 있도록 구성하였다.

덧붙이자면 Curshape 파트에 `BlockFileSave()` 넣었기에 내려오고 있던 블록의 위치는 저장되지 않고 아래에 쌓여 있던 블록의 기록만 저장된다. 이 외에도 `TetrisSave(),TetrisCall()`를 통해 점수를 저장하고 불러오는 기능을 위와 같은 매커니즘으로 비교적 간단하게 구현 할 수 있었다.

Tetris Code Review

- **tBoard.java(4)**

-게임종료 확인파트

```
private void checkGameEnd() {                       // 게임이 오버되는 상황을 체크하는 함수이다.
    int[][] coor = curShape.getcoor();              // 블럭이 생성되는것을 우리는 알수있는데 여기
    for (int row = 0; row < coor.length; row++) {   // 게임이 끝나게 하는 메서드다.
        for (int col = 0; col < coor[0].length; col++) {
            if (coor[row][col] != 0) {
                if (board[row + curShape.getY()][col + curShape.getX()] != null) {
                    state = STATE_GAME_END;
                }
            }
        }
    }
}
```

-

게임을 진행할 때 초기에 설정되는 좌표는 x=3, y=0 축이다. 게임창으로 확인하면 상단 가운데 정도가 되는데, getX(), getY() 를 통해 반환되는 초기 좌표 값을 확인해 각 블록의 배열이 초기위치에서 생성되는 블록과 교차하는지를 확인해 게임의 state를 결정짓는 중요한 메소드이다.

-상황 별 pop 문구 설정

```
if (state == STATE_GAME_END) {                  // 게임이 OVER되는 구간에 띄워질 문구를 설정하는 부분이다.
    g.setFont(new Font("Georgia", Font.BOLD, 30));
    g.setColor(Color.RED);
    g.drawString("[ GAME OVER ]", 90, 200);
    g.setFont(new Font("Georgia", Font.BOLD, 20));
    g.setColor(Color.black);
    g.drawString("DO YOU WANT  REGAME? PRESS [S] !", 20, 230);

}

if (state == STATE_GAME_STOP) {                 // 게임이 PAUSE되는 구간에 띄워질 문구를 설정하는 부분이다.

    g.setFont(new Font("Georgia", Font.BOLD, 30));
    g.setColor(Color.RED);
    g.drawString("[ GAME STOP ]", 90, 200);

    g.setFont(new Font("Georgia", Font.BOLD, 20));
    g.setColor(Color.black);
    g.drawString("Press spacebar to continue the game !", 20, 230);
}
```

Paintcomponet내의 GUI 구성중 일부분이다. 숫자로 지정해준 state의 상태에 따라 그에 맞는 문구의 폰트와 색, 글씨크기를 설정해 게임이 가져야하는 기본적인 인터페이스를 가꾼 부분이다.

Tetris Code Review

- **tBoard.java(5)**

```
// 키보드 이벤트 처리를 구현하는 part
    @Override
    public void keyPressed(KeyEvent e) {            //키보드로 작동하는 파트를
        if (e.getKeyCode() == KeyEvent.VK_DOWN) {   //회전,하,좌,우 모두 컨트
            curShape.getFaster();                   //각 파트의 메소드는 아래.
        } else if (e.getKeyCode() == KeyEvent.VK_LEFT) {
            curShape.crtleft();
        } else if (e.getKeyCode() == KeyEvent.VK_RIGHT) {
            curShape.crtRight();
        } else if (e.getKeyCode() == KeyEvent.VK_UP) {
            curShape.rotationShape();               //메소드의 구현은 다음 파'
        }

        // 게임진행중 R키를 누르면 점수와 게임판이 모두 초기화 된다.
        if (state == STATE_GAME_PLAYING) {
            if (e.getKeyCode() == KeyEvent.VK_R) {
                score = 0;                                          //
                for (int row = 0; row < board.length; row++) {
                    for (int col = 0; col < board[0].length; col++) {
                        board[row][col] = null;                     //
                    }
                }
                setCurShape();
                state = STATE_GAME_PLAYING;
            }
        }

        // 게임오버가 된후 S키를 통해 게임판과 점수판이 초기화된다.
        if (state == STATE_GAME_END) {
            if (e.getKeyCode() == KeyEvent.VK_S) {
                score = 0;                                          //
                for (int row = 0; row < board.length; row++) {      //
                    for (int col = 0; col < board[0].length; col++) {
                        board[row][col] = null;
                    }
                }
                setCurShape();
                state = STATE_GAME_PLAYING;
            }
        }

        // 게임중간에 정지를 할수있는 기능이다.
        if (e.getKeyCode() == KeyEvent.VK_SPACE) {
            if (state == STATE_GAME_PLAYING) {
                state = STATE_GAME_STOP;
            } else if (state == STATE_GAME_STOP) {
                state = STATE_GAME_PLAYING;
            }

        }

    }
```

-

사실 가장 중요한 부분이라고도 할 수 있는 **이벤트 처리 파트이다**. 처음에는 마우스를 이용한 이벤트 처리를 생각은 해보았으나 마우스로 따로 처리할 만한 기능들을 생각해내지 못해 배제했다. 테트리스 블록의 움직임은 보통 3가지로 나뉜다, 좌, 우, 아래이다. 여기서 좌우는 사실 속도 처리없이 x좌표를 기준으로 하나씩 +1 혹은 -1 처리를 해주면 되지만 아래로 이동하는 경우는 조금 다르다. 블록은 아무런 조작을 하지 않아도 일정 주기에 맞춰 아래로 떨어지게 되어있다. 고로 아래 방향키의 사용이 불필요 할 수 있지만 속도 처리를 통해 아래 키의 효율성을 부가할 수 있다. 아래의 함수를 통해 기본으로 설정된 NOMAL 속도를 키보드 입력을 통해 FASTSET 속도로 변환해 낙하 속도를 증진시킬

```
public void getFaster() {
    DELAYTIME_OPERATION = FASTEST;

}
```

수 있다. 이 외에도 게임이 종료된 후의 이벤트 처리를 통한 state 변환, 게임 중간 일시 정지 등 간단하게 설정해 놓은 state 상수를 통해 상황별 기능을 비교적 간단하게 처리할 수 있다.

Tetris Code Review

- **shping.java(1)**

-update 파트

```
public void refresh() {          // 블록을 업데이트해주는 기능할 하는구간이다.
    if (BUMPER) {                // BUMPER -> 땅에 부딪힐때마다 -> 설정해둔 Y좌표에 닿을때마다
        for (int row = 0; row < coor.length; row++) {
            for (int col = 0; col < coor[0].length; col++) {
                if (coor[row][col] != 0) {
                    board.getBoard()[y + row][x + col] = color; // 별도선언 coor 배열에 값이 없지 않다면
                                                                // 게임판을 채우기위한 컬러를 삽입해준다.

                }
            }
        }

        checkLine(); // 한줄 체크해 모두채워져있는게 확인되면 삭제하는 함수이다.
        board.addScore();   // update 될때마다 스코어가 증가하게되는 함수이다.
        board.setCurShape(); // 현재의 테트리스 도형을 설정한다.
        return;
    }
```

내용 설명에 앞서 BUMPER는 바닥에 닿는 즉, 특정 좌표와의 충돌을 검사하는 boolean이며, coor는 Color로 선언한 배열인 메인 배열, board의 정보를 정수형으로 받아들인 배열이다.

Coor의 종횡 배열 값을 탐색해 빈칸이 아닌 곳이 발견되었을 때, (0이 아니다= 색이 채워져 있다) 해당 색을 채우는 메소드다. 바닥에 블록이 바닥에 안착했을 때 기준으로 메소드는 작동한다. 이 메소드는 tBoard.java의 timer 파트에서 일정 주기로 지정된 메소드 내에서 계속 반복해 색을 채우게 되어있다. (tBoar.java 96 lines) 이후 차례로 완성된 라인은 없는지 확인, 하나씩 블록이 바닥에 닿아 누적될 때 마다 점수의 증가, 다시 새로운 블록의 생성 이 순으로 반복되게 된다.

-줄 삭제 파트

```
private void checkLine() {
    int bottom = board.getBoard().length - 1;                       //설정
    for (int top = board.getBoard().length - 1; top > 0; top--) {   //차례
        int cnt = 0;                                                //시작
        for (int col = 0; col < board.getBoard()[0].length; col++) {//그 ㄷ
            if (board.getBoard()[top][col] != null) {
                cnt++;
            }
            board.getBoard()[bottom][col] = board.getBoard()[top][col];

        }
        if (cnt < board.getBoard()[0].length) {
            bottom--;
        }
    }
}
```

테트리스의 승패를 가르는 줄 삭제 메소드다. 설정된 게임판의 y축 끝에서 누적된 블록의 일치를 확인을 하고, 차례로 한줄씩 위로 올라가며 줄의 일치 여부를 파악하게 된다. 줄이 삭제되는 것도 중요하지만 삭제된 이후에 위에 쌓여 있던 블록의 위치도 바닥으로 옮겨지도록 단계적 삽입을 진행해야 한다.

Tetris Code Review

- **shping.java(2)**

-블록의 회전처리

```
// 블록의 행과열을 반전시켜주는 구간
public int[][] transMatrix(int[][] matrix) {
    int[][] temp = new int[matrix[0].length][matrix.length];
    for (int row = 0; row < matrix.length; row++) {
        for (int col = 0; col < matrix[0].length; col++) {
            temp[col][row] = matrix[row][col];
        }
    }
    return temp;
}

public void reverseRow(int[][] matrix) {
    int mid = matrix.length / 2;
    for (int row = 0; row < mid; row++) {
        int[] temp = matrix[row];
        matrix[row] = matrix[matrix.length - row - 1];
        matrix[matrix.length - row - 1] = temp;
    }

}
```

이 부분도 상당히 신경 써서 구성한 코드 부분 중 하나다. 삽입된 블록을 행과 열을 통해 trans 하게 된다면 사실 이해할 수 없는 모양이 나타난다. 예를 들어 기역자 모양의 블록이 있다고 가정해 보자, 그렇다면 행렬의 반전을 통한 결과를 도출하는 `transMatrix(int[][] matrix)` 에서는 어떤 모양을 반환할 수 있을까?

위 와 같은 형태를 띄게 되는데 이는 회전을 통해 나올 수 있는 형태가 아니다. 뒤집는 다면 상황은 달라 질 수 있겠지만 테트리스에서 요구하는 기능은 플립이 아닌 로테이션이다. 그래서 이후 `reverseRow(int[][] matrix)`를 통해 한 번 더 정체를 거쳐 로우의 위치를 다시 이동시켜 아래의 모습을 갖출 수 있게 된다.

정사각형 모양을 블록은 사실 별도의 회전이 필요 없지만 좌우가 비대칭은 블록의 회전처리는 필수 불가결한 존재이다.

Tetris Code Review

- **shping.java(3)**

-블록 처리

```
public void render(Graphics g) {
    for (int row = 0; row < coor.length; row++) {
        for (int col = 0; col < coor[0].length; col++) {
            if (coor[row][col] != 0) {
                g.setColor(color);
                g.fillRect(col * tBoard.SET_tSPACE_SIZE + x * tBoard.SET_tSPACE_SIZE, row * tBoard.SET_tSPACE_SIZE + y * tBoard.SET_tSPACE_SIZE,
                        tBoard.SET_tSPACE_SIZE, tBoard.SET_tSPACE_SIZE);
            }

        }
    }
}
```

tBoard.java의 페인트 컴포넌트 파트에서 본 메서드다. 메서드의 설정 위치에서도 알 수 있듯이 GUI에 비춰지는 블록의 차지를 색으로 나타내주는 메서드다. 설정된 보드의 사이즈에 맞춰서 보드 사이즈내의 블록 사이즈를 고려해 다 코드에서 설정된 색을 설정하고 입히는 과정을 반복하여 우리가 봐오던 테트리스의 모습을 보여주게 된다.

-블록 회전 처리 및 예외처리

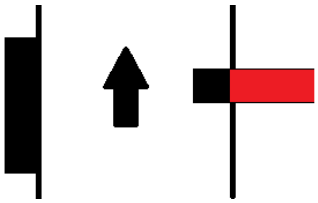
```
public void rotationShape() {
    int[][] rotatedShape = transMatrix(coor);
    reverseRow(rotatedShape); // 기존에회전되었다고 한 rotatedShape은 배열의 순서만 반대로 배열한것이기에 실제로
                              // 회전을 해서 나올수 없는 형태가 발생한다. 리버스로우를 통해 이를 바로 잡는다. 아래 구현코드를 확인하자.

    // 회전할때 생기는 예외 (회전하면서 벽과 바닥을 뚫어버리는) 확인란이다.
    if (x + rotatedShape[0].length > tBoard.SET_tBOARD_WID || (y + rotatedShape.length > tBoard.SET_tBOARD_HEI)) {
        return;
    }

    // 회전중 다른 모양과 충돌하여 벽과 마찬가지로 도형을 뚫어버리는걸 막는, 확인하는 란이다.
    for (int row = 0; row < rotatedShape.length; row++) {           //회전을 하면서 기존이 존재하는 블록의 가로세로는
        for (int col = 0; col < rotatedShape[row].length; col++) {  //다르기때문에 블록이 어느위치에서 회전을 했냐에따라
            if (rotatedShape[row][col] != 0) {                      //필요로하는 공간이 다르다.
                if (board.getBoard()[y + row][x + col] != null) {
                    return;
                }
            }
        }
    }

    coor = rotatedShape;
}
```

구성된 코드를 자세히 보면 위에서 살펴봤던 메서드를이 내부에 구성되어 있는 것을 확인할 수 있다. 블록이 두번의 메소드를 거쳐 회전을 진행할 수가 있지만 언제나 회전할 수 있지는 않다. 특정 좌표, 벽과 밀착 (제한된 x좌표와 근접된 상태)상태에서의 회전은 사실 불가능하다. 하지만 이를 처리하지 않으면 GUI상에서는 게임판을 벗어나는 현상이 초래된다. 회전을 통해 바뀐 가로, 세로 즉 행과열의 범위를 비교해 기존에 있었던 위치에서의 범위 초과여부를 확인해 회전이 가능하게, 혹은 불가능하게 설정을 만져준다.



위와 같은 상황을 처리하기위한 구현부라고 말 할 수 있다.

게임 방식 설명 및 시연

- 게임 방식 설명

게임의 방법은 다음과 같다. 게임을 실행한 후 블록의 이동은 방향키로 조작하고, 블록의 회전은 UP 방향키로 진행한다. 게임 도중 일시정지를 원할 때는 SPACEBAR를 누르면 게임이 일시 정지한다. 이후 다시 문구에 따라 SPACEBAR를 누르면 다시 게임이 그 상태에서 실행된다. 블록의 초과로 게임이 끝나면 재시작을 하겠냐는 문구와 함께 게임이 종료되는데 이때 S키를 누르면 점수와 블록이 초기화되고 게임이 재시작 된다. 게임이 종료된 이후가 아니라 게임 도중 재시작을 원한다면 R키를 누르면 된다. 그럼 재시작과 마찬가지로 점수와 블록이 초기화 된 후 게임이 진행된다. 점수의 상승의 기준은 누적된 블록의 수만큼 정량적으로 증가한다.

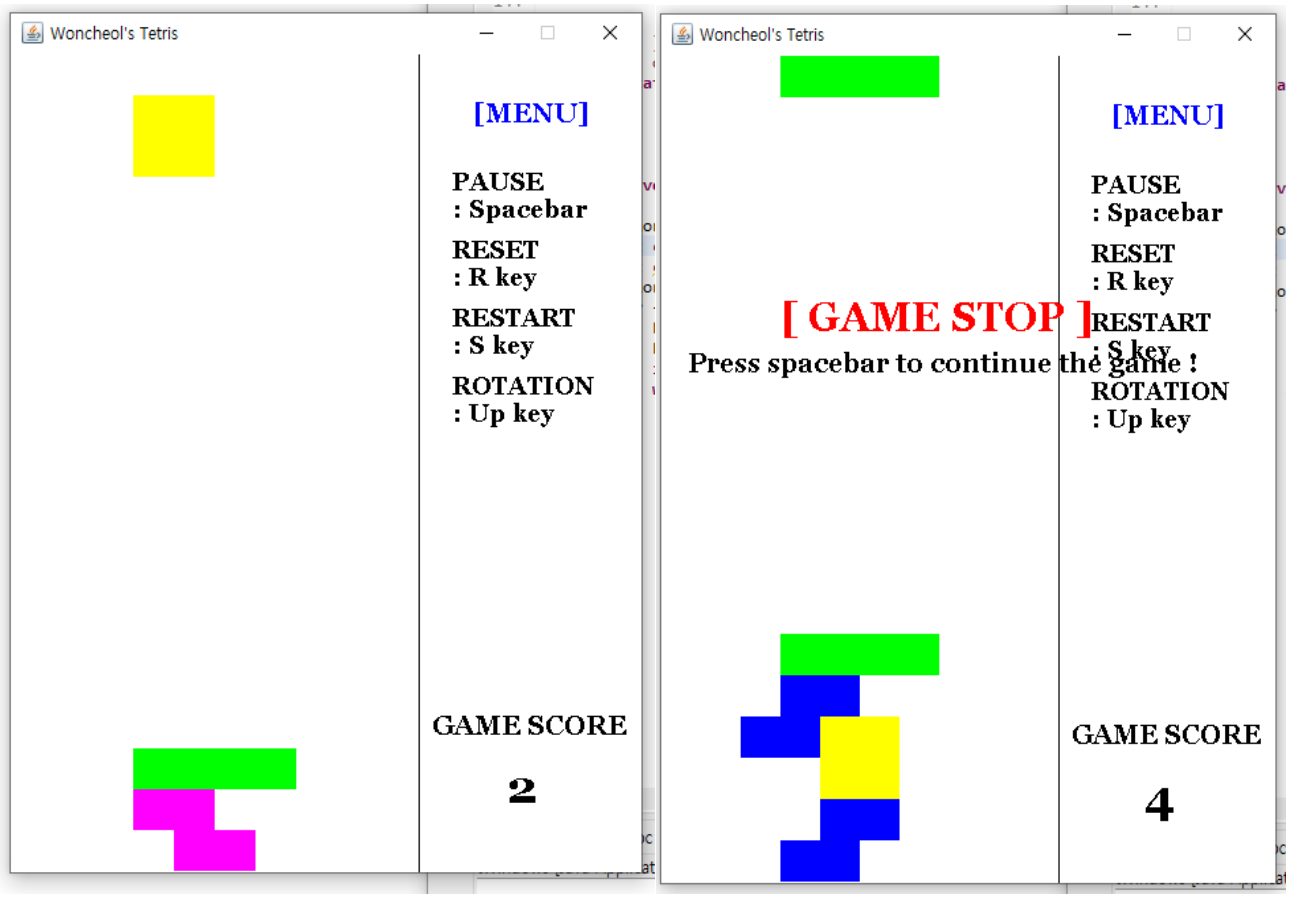
게임도중 물리적으로 게임을 종료하면 종료한 시기에 기록된 블록만큼 설정한 파일경로에 블록 위치가 기록 된다. 이후 다시 게임을 시작하면 이전에 진행했던 게임상황을 불러와 이어서 게임을 진행할 수 있다.

게임점수 저장 경로: `D:\\Tetris.txt`

블록위치 저장 경로: `D:\\Tetris1.txt`

게임 방식 설명 및 시연

- 게임 시연

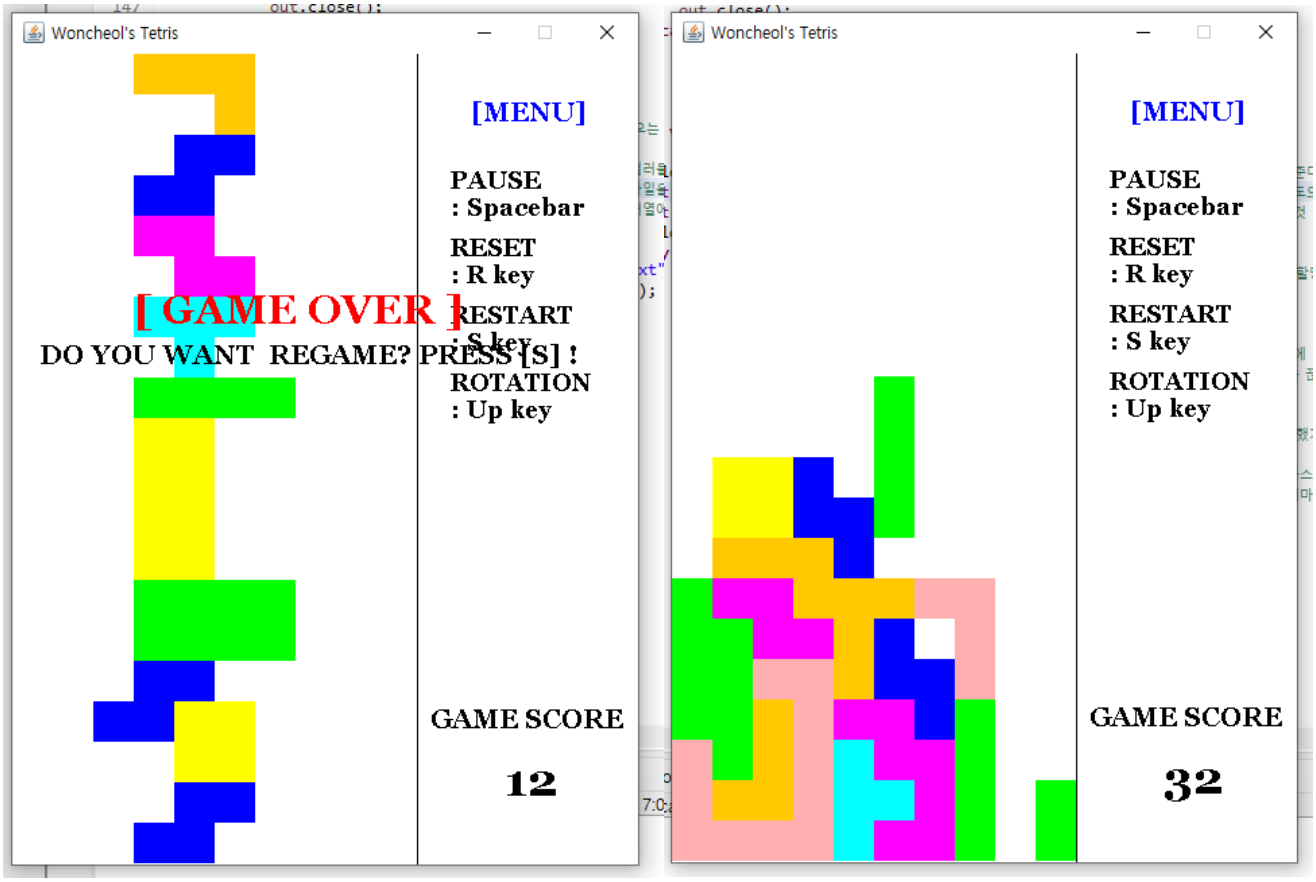


그림(좌)의 모습은 게임을 실행한 이후의 초반부 모습이다. 전반적인 인터페이스는 우측에는 단축키를 알려주는 메뉴바와 스코어보드로 구성 되어있다. 그림에서 볼 수 있듯이 누적된 블록의 수만큼 점수가 측정되는 것을 확인할 수 있다.

그림(우)는 SPACE바를 통한 STOP, PAUSE 기능이다. 해당키를 입력할 시 블록의 모든 이동이 멈추게 되며 경고 문구에 따라 다시 SPACE를 누르게 되면 게임이 곧 바로 진행된다.

게임 방식 설명 및 시연

- 게임 시연



그림(좌)는 누적된 블록이 지정 범위를 벗어나게 되어 게임이 종료된 모습이다. 종료됨과 동시에 경고 문구와 함께 재시작을 물어보는데 여기서 S키를 누르게 되면 점수와 블록의 초기화와 함께 게임이 재시작 되게 된다. 여기서 중요한점은 S키는 종료 후 재시작 키이며, R은 언제든 게임 진행중 사용할 수 있는 재시작 키 라는 것이다.

그림(우)는 정상적인 게임진행 모습을 보인다. 저렇게 게임상에서 한 번에 많은 블록을 삭제할 수 있는 조건에서의 줄삭제도 원활히 이루어진다.

-평가 지표확인

기본 평가 지표	구현여부	구현한 추가 기능	구현 여부
블록의 이동	o	게임 일시 정지	o
블록 한 줄 삭제	o	게임 중 재시작	o
파일 저장	o	게임종료 후 재시작	o
파일 가져오기	o	스코어 보드	o
블록 랜덤시작, 블록 낙하	o		
회전	o		
게임종료, 프로그램 종료	o		

Assignment review

- **과제를 하며 느낀 점**

이번 테트리스 과제를 통해 느낀 점이 상당히 많다. 경제학을 전공하던 전과생으로 사실 프로젝트 과제를 받아본적이 없었다. 늘 발전을 위해 공부를 해왔었지만 이번 테트리스 과제는 나에게 상당히 버거운 과제임이 틀림없었다. 그래서 넉넉하게 준 기간임에도 불구하고 남들보다 빠르게 과제에 먼저 손을 댔던 것 같다. 막무가내로 어떻게 구성할지 코드부터 작성하는 것이 아니라, 어떻게 만들지 구상부터 하는게 중요 하다는 것을 알았다. 처음엔 손을 댈 수가 없었다. 그래서 사실 코딩이지만 컴퓨터가 없어도 될 정도로 키보드에 손에 가지 않았다. 그래서 내가 구상하는 가상의 테트리스 화면을 종이에 담았고, 가상의 기능도 넣어봤다. 이 후에 조금씩 내가 과제를 바라보는 시선이 달라 졌던 것 같다. 원하는 사이즈의 프레임을 띄우고, 타이틀을 붙여 넣고, 게임을 진행하는 영역과, 문구를 넣는 영역을 차례로 나누고 이런 사소한 과정을 진행하면서 답이 서서히 보이기 시작했다.

하지만 좋은 코드는 어느 누가 봐도 빠르게 이해할 수 있고, 보수할 수 있는 코드라고 생각하는데 내 코드는 그렇지 못했다. 사전에 공지한 필수 요건 중 하나인 파일 입출력에 관련한 부분을 고려하지 않고 틀을 짰기 때문에 사실 파일을 읽고, 써오는 것이 여간 힘든 게 아니었다. 또한 나날이 과제코드를 수정하고 작성한 날은 괜찮았지만, 격일로 과제에 손을 댔을 때는 코드정보 주석을 남기는 습관을 들이지 않아서 다시 코드를 이해하는데 시간을 배로 쓰는 수모를 겪었다. 이러한 과정을 통해서 기록하는 습관이 얼마나 중요한지 몸소 체험했고, 내가 생각해서 작성한 코드라고 늘 그 로직이나 형태가 내 머리에 계속 남아있는 것은 아니라는 것을 느꼈다.

이후 그렇게 내가 구성한 프로그램으로 어느 정도의 게임을 할 수 있는 구색이 갖춰졌을 때, 그 뿌듯함 이로 말할 수 없었다. 기본에 충실한 테트리스를 만들자는 것이 1차적인 목표였기 때문에 외형적으로는 볼품없었지만 내가 무언가 움직이는 것을 만들어 이뤄냈다는 생각에 사실 의지는 더 불타올랐다. 그렇게 다음으론 자잘한 게임상의 오류나 예외를 파악하고 수정해 나갔고, 기능적인 부분뿐만 아니라 외형적인 부분에도 신경을 써서 게임을 다듬어 나갔다. 나름 멋지게 만든다고 했지만 다른 기성 게임들처럼 만들기에는 역부족이었던 것 같다.

이번 과제는 내가 대학을 다니면서 들인 시간 중에 가장 많은 시간을 쏟은 과제인 것 같다. 사실 지금 와서 작성해 놓은 코드들을 살펴보면 그렇게 복잡하고 어려운 로직들은 없는 것 같다. 프로그래밍이 조금 익숙한 학생들에겐 며칠이면 뚝딱 만들 수 있을 정도로 생각이 되기는 하지만 부족한 나에겐 계속 공부해야 하는 부분이 존재했고, 그 과정속에서는 어려운 부분이 많았다. 하지만 남과의 비교는 끝이 없고 열등감만 불러 일으킨다. 일단은 과거의 나보다 오늘의 "나" 가 더욱 발전했다는 것에 보람을 느낀다. 그렇게 과제를 마치고 떠오르는 생각은 "이렇게 오래 열심히 했으니 이제 그만하고 싶다." 라는 생각보다, 내가 이렇게 해낸 걸 증명했으니 더 공부해보고 싶다는 생각이 종강을 앞두고 막 솟아나기 시작했다. C++을 공부하면서는 느껴 본적이 없는 감정이라 자바에 대한 매력이 더욱 더 크게 느껴지는 부분이었다.

이번 학기에 자바수업을 듣게 되어서 C++이외의 언어를 처음 배우게 되는 기회가 됐는데, 이렇게 자바에 흥미를 갖게 된 기회가 된 것 같아 정말 후회 없는 선택이었던 것 같다. 앞으로 자바도 열심히 해보겠지만 교수님의 조언에 따라 다른 다양한 언어를 접해보면서 공부해보는게 내가 앞으로 진행해 봐야할 1차적인 목표가 아닐까 싶다. 비록 온라인 수업이었지만 여러가지로 얻어가는게 많았던 수업이었던 것 같다.

감사합니다.