자바기반응용프로그래밍

기말과제(테트리스) 보고서

이름 : 박기범

학번 : 12191601

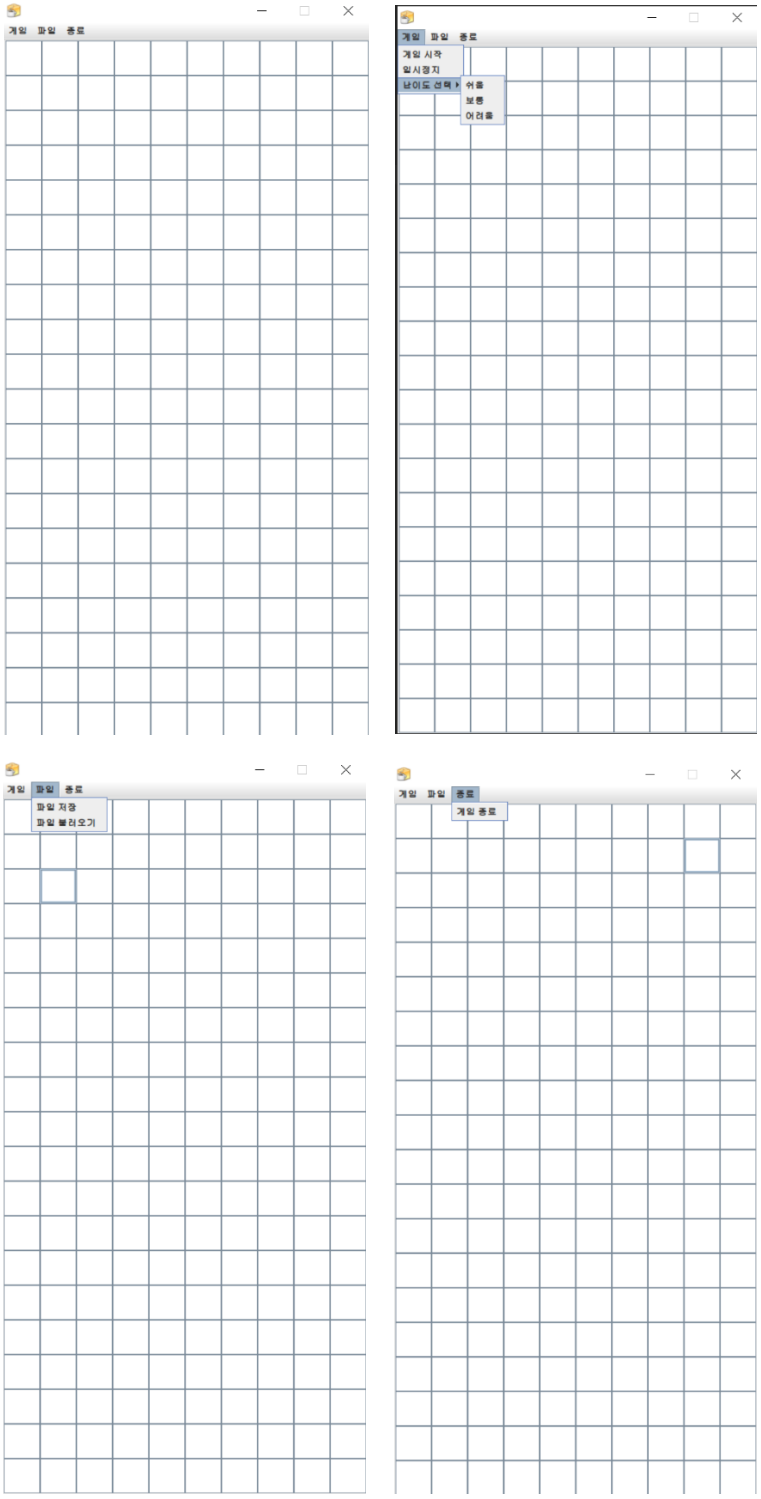
담당교수 : 임광수 교수님

분반 : 001

학과 : 컴퓨터공학과

< 목차 >

1. GUI 화면 구성 및 프로그램 기능 설명



프로그램을 실행하면 기본적으로 나오는 GUI화면이다. 20 X 10의 배경으로 이루어져 있으며 메뉴는 게임, 파일, 종료로 구분되어 있다.

처음 있는 '게임' 메뉴에는 총 3가지 메뉴옵션이 있다. '게임 시작', '일시정지', '난이도 선택' 메뉴가 있다. '게임 시작'은 이름과 같이 게임을 시작하는 역할을 한다. 이 메뉴를 선택할 경우

gameStart()함수가 실행되면서 쓰레드가 동작한다. 또한 일시정지시킨 게임을 다시 시작시키는 역할도 한다. '일시정지'는 게임을 일시정지시키는 버튼이다. 동작하는 쓰레드 내부에 boolean타입 변수를 활용해서 중간에 무한루프를 돌리게 하는 역할을 한다. '난이도 선택'은 총 3가지 난이도를 선택해서 게임 스피드를 조절할 수 있다. 게임 스피드를 조절하는 기준은 블록이 떨어지는 속도와 블록의 모양 개수이다. 떨어지는 스피드는 speed 변수를 활용해서 Thread.sleep(speed)로 시간을 조절했다. 또한 maxBlock이라는 변수를 활용해서 '쉬움'난이도를 선택할 경우 블록이 만들어지는 종류를 4개로 조절했다. 나머지 난이도는 모두 6개 종류의 블록이 나온다.

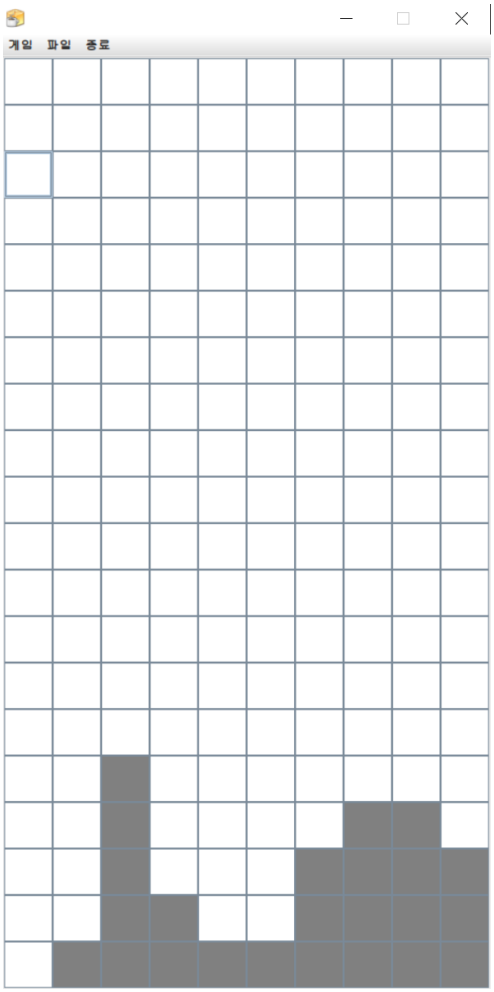


그림 1 '파일 불러오기' 를 통해 불러온 예시

'파일'메뉴의 메뉴옵션은 '파일 저장', '파일 불러오기'가 있다. '파일 저장'메뉴는 블록이 바닥에 닿았을 경우 배경에 고정되는 형태의 현재 상태를 메모장에 텍스트파일로 저장시키는 역할을 한다. 후에 서술할 코드 설명에서 어떤 형식으로 해당 데이터를 저장하는 지 자세히 설명하겠지만 boolean타입의 배경 배열의 내용을 저장한다. '파일 불러오기'는 '파일 저장'에서 저장한 save.txt파일을 읽어와서 다시 기존의 배경배열에 저장한다.

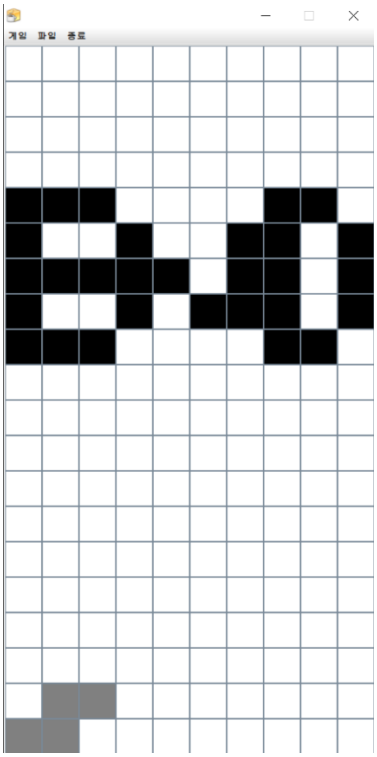


그림 2 '게임 종료' 를 누를 경우 나오는 화면

마지막으로 '종료'메뉴의 '게임 종료'는 이름 그대로 게임을 종료하는 역할을 한다. 쓰레드 내부의 while문을 돌리는 변수를 정지시키고 interrupt를 통해서 쓰레드를 정지시킨다.

기본 조작은 space바는 블록 내리기, 좌우키는 좌우 이동, 위아래키는 블록회전이다.

2. 코드 설명

이 부분은 함수에 사용된 변수화 함수들을 설명하는 부분이다. 단순 대입 반복문과 단순히 버튼 색을 채우는 역할을 하는 함수는 설명을 생략하겠다.

2-1. 변수 설명

```
static final Color BACKGROUND = Color.WHITE;
static final Color FIXED = Color.GRAY;
boolean fixedBoard[][] = new boolean[20][10];
int rotation;
int rotClick;
JButton board[][];
Color colorList[] = {Color.PINK, Color.CYAN, Color.BLUE, Color.BLACK, Color.RED, Color.ORANGE, Color.MAGENTA, Color.YELLOW};
Thread realtime;

int speed = 1000;
int maxBlock = 6;
int score = -1;
int block; // curr block type
boolean pause = false;
boolean reset = true; // reset flag
boolean gameend = false; // gameend flag
```

우선 가장 기본적으로 활용하는 변수들이다. BACKGROUND와 FIXED는 버튼 색을 칠하는 경우에 사용하기위한 final Color변수이다.

fixedBoard는 테트리스 블록이 최하단에 도달했을 경우 블록이 고정되는데, 이 고정되는 부분을 표시, 기록해주는 변수이다. '파일 저장'버튼을 누를 경우 fixedBoard에 있는 부분을 1과 0으로 텍스트 파일에 기록하게 된다. 그리고 '파일 불러오기'를 했을 때 텍스트 파일에 저장된 값을 fixedBoard에 기록한다.

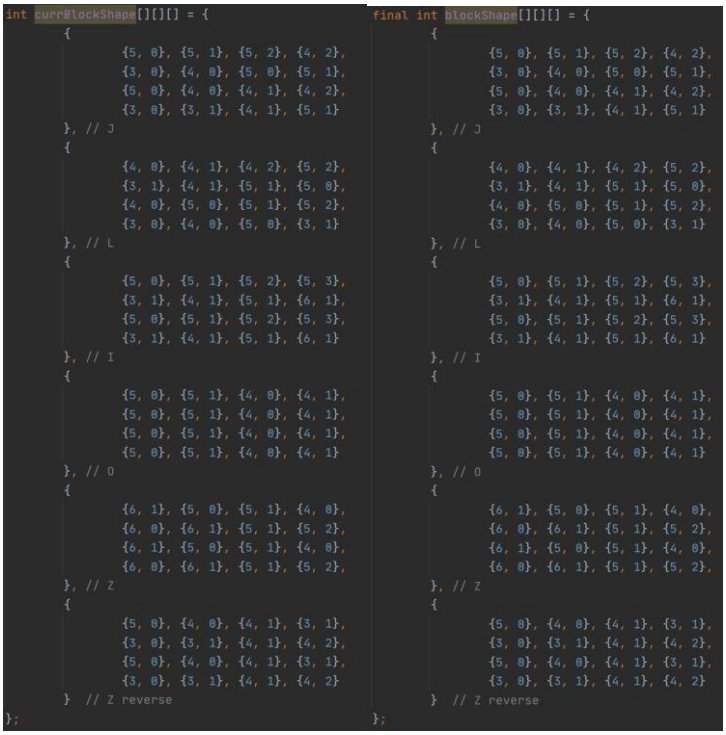
rotation은 현재 블록의 회전상태를 저장하는 변수이고 board는 JButton의 색을 저장하는 역할을 한다. 전체 배경색과 만들어지는 테트리스 블록의 모양을 GUI에 표시해주는 역할을 한다.

colorList변수는 블록의 색상 값을 정해주는 역할을 한다. 이후에 랜덤 값으로 색을 가져온다.

realtime 쓰레드 변수는 이 프로그램에서 돌아가는 쓰레드 정보를 저장한다. 이 변수로 interrupt와 start를 다룬다.

speed변수, maxBlock은 게임의 난이도를 결정하는 변수들이다. Thread의 sleep시간을 결정해서 블록이 내려오는 속도를 결정하는 역할을 하는 것이 speed, 블록의 랜덤 값을 조정해서 게임에서 나오는 블록의 종류를 결정하는 역할을 하는 변수가 maxBlock변수다. block은 maxBlock을 통해서 결정되는 블록의 랜덤 값을 지정해준다. 이 값을 통해서 어떤 블록이 형성되어 만들어지는 지를 결정해준다.

pause, reset, gameend변수는 쓰레드 내부에서 돌아가는 boolean타입의 변수이다. pause는 말 그대로 일시정지 상태를 결정한다. pause변수의 상태에 따라 비어 있는 while문을 통해서 무한루프를 돌게 한다. reset은 블록을 새로 만들어주는 함수를 실행시키는 여부를 결정한다. gameend는 쓰레드 내부에서 도는 전체 프로그램을 돌리는 while문을 돌아가게 하는 역할을 결정한다.



currBlockShape변수는 현재 내려오는 블록의 이동상태를 기록한다. 즉 블록이 내려오거나 좌우로 이동하는 좌표는 이 변수에 실시간으로 기록한다. 한마디로 말하면 이 변수가 곧 블록의 이동형태를 결정한다. 주석에 적혀진 블록이 나왔을 때, 그 블록의 형태로 버튼의 색을 칠하는 좌표를 나타낸다. 총 4개의 줄로 적은 이유는 각 줄의 좌표들은 반시계방향으로 90도 회전시켰을 때 나타나는 블록의 모양을 버튼에 색을 칠해주는 좌표를 저장한다. blockShape변수는 final로 저장되어서 블록의 초기 좌표를 기록한다. 즉 현재 내려오는 블록의 위치를 다시 초기 위치로 돌려주는 역할을 한다.

2-2. Source() : 생성자

```
Source() {
    Init();
    MakeMenu();

    setSize( width: 500,  height: 1000);
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    setResizable(false);
    setVisible(true);

    addKeyListener( l: this);
    setFocusable(true);
    setFocusTraversalKeysEnabled(false);
}
```

GUI를 구성하는 역할을 한다. Init()함수는 기본적인 테트리스 GUI를 형성하는 역할을 한다. 그

외에는 Size와 프로그램 닫기 버튼 활성화와 같은 기본적인 GUI세팅을 다루는 코드들이다. 이때, 버튼의 외관상 디자인 때문에 GUI화면 크기를 조절할 수 없게 setResizable을 false로 설정해서 화면 크기를 조절할 수 없게 설정했다. 그 외에는 키를 누를 때 결정하는 옵션들을 나타낸다.

2-3. Init(), MakeMenu() 함수 : 테트리스 초기화, GUI에 메뉴 삽입 함수

```
void Init() {
    board = new JButton[20][10];

    setLayout(new GridLayout( rows: 20,  cols: 10));
    for (int i = 0; i < 20; i++) {
        for (int j = 0; j < 10; j++) {
            board[i][j] = new JButton();
            add(board[i][j]);
            board[i][j].setBackground(BACKGROUND);
        }
    }
}
```

```
void MakeMenu() {
    JMenuItem item = null;
    JMenuBar mb = new JMenuBar();
    JMenu m1 = new JMenu( s: "게임");
    JMenu m2 = new JMenu( s: "파일");
    JMenu m3 = new JMenu( s: "종료");

    AddMenu(item, m1,  MenuName: "게임 시작");
    AddMenu(item, m1,  MenuName: "일시정지");

    JMenu subMenu = new JMenu( s: "난이도 선택");
    AddMenu(item, subMenu,  MenuName: "쉬움");
    AddMenu(item, subMenu,  MenuName: "보통");
    AddMenu(item, subMenu,  MenuName: "어려움");
    m1.add(subMenu);

    AddMenu(item, m3,  MenuName: "게임 종료");

    AddMenu(item, m2,  MenuName: "파일 저장");
    AddMenu(item, m2,  MenuName: "파일 불러오기");

    mb.add(m1);
    mb.add(m2);
    mb.add(m3);

    setJMenuBar(mb);
}

void AddMenu(JMenuItem item, JMenu m, String MenuName) {
    item = new JMenuItem(MenuName);
    item.addActionListener( l: this);
    m.add(item);
}
```

우선 가장 먼저 실행되는 Init()함수이다. 간단하게 JButton의 색을 모두 흰색으로 칠해주는 역할과 JButton을 생성하고 GridLayout을 설정하는 역할을 한다. 또한 MakeMenu는 메뉴 상태를 결정해주는 역할이다. 반복작업을 해야 하기 때문에 AddMenu라는 함수로 메뉴관련 설정들을 설정했다.

2-4. actionPerformed(ActionEvent e) 함수 : 메뉴관련 실행 함수

```java
@Override
public void actionPerformed(ActionEvent e) {
    JMenuItem value = (JMenuItem) e.getSource();

    if (value.getText().equals("게임 시작")) {
        if (score == -1) score = 0;
        pause = false;
        gameStart();
    }else if (value.getText().equals("게임 종료")) {
        gameEnd();
        realtime.interrupt();
        endPrint();
    }else if (value.getText().equals("파일 저장")) {
        try {
            save();
        } catch (IOException ioException) {
            ioException.printStackTrace();
        }
    }else if (value.getText().equals("파일 불러오기")) {
        load();
        for (int i = 0; i < 20; i++){
            for (int j = 0; j < 10; j++) {
                if (fixedBoard[i][j]) board[i][j].setBackground(FIXED);
            }
        }
    }else if (value.getText().equals("쉬움")) {
        speed = 1000;
        maxBlock = 4;
    }else if (value.getText().equals("중간")) {
        speed = 600;
        maxBlock = 6;
    }else if (value.getText().equals("어려움")) {
        speed = 300;
        maxBlock = 6;
    }else if (value.getText().equals("일시정지")) {
        pause = true;
    }
}
```

각 메뉴 별로 사용된 함수의 역할을 간단하게 설명해보면, 우선 '게임 시작' 메뉴는 일시정지된 게임을 다시 실행하는 역할도 하기 때문에 pause변수를 false로 돌려서 일시정지를 해제한다. 그 후 gameStart()함수로 쓰레드를 실행한다.

'게임 종료' 메뉴는 gameEnd()함수를 실행해서 쓰레드 내부의 반복문을 종료시키는 역할을 한다. 이곳에 적힌 interrupt코드는 gameEnd()함수로 옮겨졌다. endPrint()는 그림2에 나온 END를 기록해주는 역할을 하는 단순한 색칠 함수이다.

'파일 저장'과 '파일 불러오기'는 각각 텍스트 파일을 쓰고 읽는 역할을 한다. '파일 불러오기'를 할 경우는 읽어 온 텍스트 기록을 저장한 fixedBoard의 상태로 배경, 즉 board변수에 FIXED로 색을 칠해준다. 난이도 관련된 설정을 누르면 난이도와 관련된 변수들 값을 지정해준다. '일시정지'

는 일시정지를 결정하는 pause변수를 true로 만들어준다. pause변수를 true로 만들면 쓰레드 내부의 빈 while문을 돌아가게 만들어준다.

2-5. gameStart(), gameEnd() 함수 : 게임 실행과 종료 관련 함수

```
void gameStart() {
    gameend = false;
    reset = true;
    realtime = new Thread() {
        @Override
        public void run() {
            int x = 1;
            while (!gameend) {
                while (pause){}

                try {
                    Thread.sleep(speed);
                      System.out.println(x);

                    if (reset) {
                        makeBlock();
                        Thread.sleep(speed);
                    }

                    blockDown();
                } catch (InterruptedException e) {

                }
            }
        }
    };
```

```
void gameEnd() {
    gameend = true;
    System.out.println("게임 종료!");

    for (int i = 0; i < 20; i++) {
        for (int j = 0; j < 10; j++) {
            if (!fixedBoard[i][j]) board[i][j].setBackground(BACKGROUND);
        }
    }
}
```

gameStart()함수는 쓰레드를 돌려주는 역할을 한다. reset변수가 true라면 블록을 새로 만드는 함수를 실행시키고, 그렇지 않다면 현재 존재하는 블록을 아래로 내려주는 역할을 하는 blockDown()함수를 실행시켜준다. gameEnd는 gameStart내부에 있는 첫번째 while문의 루프 변수인 gameend를 true로 변경해서 while문을 종료시킨다. 그 후 만약 현재 내려오고 있던 블록이 있다면 없애도록 fixedBorard가 true가 아니면 모두 배경색으로 칠한다. gameEnd()함수에 코드가 마지막에 추가되었는데 캡쳐를 못했다. 가장 마지막에 realtime.interrupt()가 추가되었다. 메뉴를 눌렀을 때 넣어둔 realtime.interrupt()를 이곳으로 옮겼다.

2-6. makeBlock(), blockDown() 함수 : 새로운 블록 만드는 함수, 블록은 내려주는 함수

```
void makeBlock() {
    this.rotation = 0;
    this.rotClick = 0;

    Random rand = new Random();
    block = rand.nextInt(maxBlock);
    int coloridx = rand.nextInt( bound: 8);

    for (int i = 0; i < 4; i++) {
        int x = currBlockShape[block][i][0];
        int y = currBlockShape[block][i][1];

        board[y][x].setBackground(colorList[coloridx]);
    }

    reset = false;
}
```

```
void blockDown() {
    if (columnFull()) gameEnd();

    Color curr = null;

    if (!isCollied( x: 0,  y: 1, rotation)) {
        for (int i = 0; i < 4; i++) {
            int currx = currBlockShape[block][i + rotation][0];
            int curry = currBlockShape[block][i + rotation][1];
            curr = board[curry][currx].getBackground();
        }

        for (int i = 0; i < 16; i++)
            currBlockShape[block][i][1]++;

        filleWhite();

        for (int i = 0; i < 4; i++) {
            int x = currBlockShape[block][i + rotation][0];
            int y = currBlockShape[block][i + rotation][1];

            board[y][x].setBackground(curr);
        }
    }else{
        boardFixed();
    }
}
```

블록을 만들 때는 모든 조건을 초기화해야 한다. 그래서 블록의 현재 회전상태도 기존상태로 돌려놓는다. 그 후 Random 변수를 통해 block의 유형 개수와 색칠할 블록의 색의 인덱스를 랜덤으로 가져온다. 그 후 0~3까지 인덱스에 저장된 기본 블록형태의 x와 y좌표들을 가져와서 해당 위치의 버튼 색을 칠한다. 그 후 reset을 false로 만들어서 현재 생성된 블록이 고정되기 전까지 새로운 블록을 만들지 않게 한다.

blockDown()함수의 경우 우선적으로 확인해야하는 것은 어떤 열이라도 꽉 찬 부분이 있다면 게임을 종료해야 한다. 그 후 만약 꽉 찬 열이 없다면 y좌표를 이동했을 때 다음 열에 현재 블록이 충돌이 발생하는 경우를 찾는다. 이 경우에 사용되는 함수가 isCollied(int x, int y, int rot)이다. 아래에 이미 고정된 블록이 있거나 더 이상 내려갈 수 없는 바닥인 경우가 아니면 현재 블록의 색을 저장하고 다음 현재 블록들의 모든 회전 형태를 갖고 있는 currBlockShape의 y좌표를 1씩 증가한다. fillWhite()함수로 fixedBoard에 true가 아닌 버튼 위치는 배경색으로 모두 채워준다. 그 후 블록이 이동한 좌표로 색을 다시 채워준다. 여기서 fillWhite()는 단순히 반복문으로 색을 채워주는 함수이다.

만약에 충돌이 발생한다면 boardFixed()함수를 불러와서 해당 블록을 고정시켜주는 역할을 한다. 자세한 설명은 boardFixed()함수 설명에서 설명하겠다.

2-7. boardFixed(), isCollied(int x, int y, int rot) 함수

```
void boardFixed() {
    for (int i = 0; i < 4; i++) {
        int x = currBlockShape[block][i + rotation][0];
        int y = currBlockShape[block][i + rotation][1];

        fixedBoard[y][x] = true;
    }

    for (int i = 0; i < 20; i++) {
        for (int j = 0; j < 10; j++) {
            if (fixedBoard[i][j]) board[i][j].setBackground(FIXED);
        }
    }

    rowDelete();
    reset = true;
    currBlockInit(); // block point reset
}
```

```
boolean isCollied(int x, int y, int rot) {
    for (int i = 0; i < 4; i++) {
        int currx = currBlockShape[block][i + rot][0];
        int curry = currBlockShape[block][i + rot][1];
        int nx = currx + x;
        int ny = curry + y;

        if (nx > 9 || nx < 0) return true;
        if (ny > 19 || ny < 0) return true;
        if (fixedBoard[ny][nx]) return true;
    }

    return false;
}
```

고정시키는 블록의 위치의 좌표들을 가져온다. 그 좌표들에 해당하는 fixedBoard의 좌표를 true로 바꿔준다. 그렇게 true로 바뀐 부분의 좌표를 확인하면서 해당하는 부분의 board좌표의 버튼의 색을 FIXED로 채워준다. 모든 블록을 고정시킨 후 rowDelete()함수로 지워져야 할 줄이 존재하는지 확인해보고 존재한다면 해당 행을 제거해준다. 그리고 reset을 true로 변경해서 makeBlock() 함수가 실행되어 새로운 블록이 나올 수 있게 해준다. 마지막으로 currBlockInit()함수를 활용해서 현재 고정시킨 블록에 해당하는 currBlockShape변수들을 모두 원래 상태로 돌려준다. 이때 사용하는 것이 blockShape변수이다.

isCollied함수는 우선 parameter로 x와 y의 좌표로 얼만큼 이동하는 지를 받아오고 회전 값이 존재하는 지를 받아온다. 이 경우는 다음에 이동하는 블록의 위치가 충돌이 일어나는 지를 확인하는 작업을 진행한다. x좌표가 양끝에 다다르거나 y좌표가 맨 밑에 도달하면 true를 반환한다. 또한 fixedBoard, 즉 고정된 블록이 존재하는 지 확인을 한 후 블록의 다음 위치에 블록이 고정된 위치라면 true를 반환해준다.

2-8. columnFull() 함수 : 열 확인 함수

```
boolean columnFull() {
    for (int i = 0; i < 10; i++) {
        if (fixedBoard[0][i]) return true;
    }

    return false;
}
```

사실 열이 꽉 찼는지를 확인하는 것보다 0번째 행이 채워져 있다면 그 앞선 행을 통해서 블록이 이미 끝까지 도달했다는 의미가 된다. 그래서 0행의 모든 열을 확인하다가 fixedBoard가 true인 부분만 찾으면 된다.

2-9. rowDelete() 함수 : 꽉 찬 행 확인 및 제거 함수

```
void rowDelete() {
    int lastdel = 0;
    int delblock = 0;

    for (int i = 19; i > -1; i--) {
        boolean rowCheck = true;

        for (int j = 0; j < 10; j++) {
            if (!fixedBoard[i][j]) {
                rowCheck = false;
                break;
            }
        }

        if (rowCheck) {
            // rowCheck true -> row delete
            for (int j = 0; j < 10; j++) {
                board[i][j].setBackground(BACKGROUND);
                fixedBoard[i][j] = false;
                score++;
            }
            delblock++;
            lastdel = i;
        }
    }

    for (int i = lastdel - 1; i > -1; i--) {
        for (int j = 0; j < 10; j++) {
            fixedBoard[i + delblock][j] = fixedBoard[i][j];
            fixedBoard[i][j] = false;
        }
    }

    for (int i = 0; i < 20; i++) {
        for (int j = 0; j < 10; j++) {
            if (fixedBoard[i][j]) board[i][j].setBackground(FIXED);
        }
    }
}
```

행을 제거하고 그 위에 존재하는 행의 블록 데이터를 밑으로 내리는 작업을 구상하는 게 가장 어려웠다. 우선 19행부터 0행까지 역으로 행을 체크하면서 확인한다. 만약 해당 행의 열의 fixedBoard에 하나라도 차 있지 않은 곳이 있다면 rowCheck변수를 false로 설정한다. 만약 rowCheck가 true면 해당 열이 제거 대상이 되는 행이라는 의미가 된다. 해당 행에 한해서 모든 열의 색을 BACKGROUND로 채워주고 fixedBoard변수는 false로 바꾼다. 또한 삭제되는 행이 몇 개인지 체크하고 마지막으로 제거된 행이 몇 행인지 저장해준다.

그 후 마지막에 제거된 행 바로 위부터 0행까지 역으로 fixedBoard데이터를 삭제된 행 개수만큼 아래로 옮겨준다. 그리고 옮긴 위치의 fixedBoard는 false로 변경해준다.

그리고 마지막으로 다시 기존에 고정위치를 채워주면 고정되어있던 행들이 이동하는 형태가 나타난다. 이때 약간의 딜레이가 발생한다.

2-10. keyPressed(KeyEvent e)함수 : 키 입력함수

```java
@Override
public void keyPressed(KeyEvent e) {
    switch (e.getKeyCode()) {
        case KeyEvent.VK_RIGHT:
            System.out.println("Right");
            move( point: +1);
            break;

        case KeyEvent.VK_LEFT:
            System.out.println("Left");
            move( point: -1);
            break;

        case KeyEvent.VK_SPACE:
            blockDown();
            break;

        case KeyEvent.VK_UP:
            System.out.println("Up");
            rotate( direction: 1);
            break;

        case KeyEvent.VK_DOWN:
            System.out.println("Down");
            score++;
            rotate( direction: -1);
            break;
    }
}
```

오른쪽과 왼쪽 키를 누르면 각각 방향으로 블록이 이동한다. 위 버튼은 반시계, 아래 버튼은 시계 방향으로 블록을 회전시킨다. 그리고 space바는 블록을 아래로 이동한다.

2-11. move(int point), rotate(int direction) 함수

```java
void move(int point) {
    Color curr = null;

    if (!isCollied(point,  y: 0, rotation)) {
        for (int i = 0; i < 4; i++) {
            int currx = currBlockShape[block][i + rotation][0];
            int curry = currBlockShape[block][i + rotation][1];

            curr = board[curry][currx].getBackground();
        }

        for (int i = 0; i < 16; i++)
            currBlockShape[block][i][0] += point;

        filleWhite();

        for (int i = 0; i < 4; i++) {
            int x = currBlockShape[block][i + rotation][0];
            int y = currBlockShape[block][i + rotation][1];

            board[y][x].setBackground(curr);
        }
    }
}
```

```java
void rotate(int direction) {
    int newRotation = rotation + 4 * direction;

    if (newRotation > 12) newRotation = 0;
    if (newRotation < 0 ) newRotation = 12;

    if (!isCollied( x: 0,  y: 0, newRotation)) {
        Color curr = null;
        for (int i = 0; i < 4; i++) {
            int currx = currBlockShape[block][i + rotation][0];
            int curry = currBlockShape[block][i + rotation][1];

            curr = board[curry][currx].getBackground();
        }

        rotation = newRotation;
        filleWhite();

        for (int i = 0; i < 4; i++) {
            int x = currBlockShape[block][i + rotation][0];
            int y = currBlockShape[block][i + rotation][1];

            board[y][x].setBackground(curr);
        }
    }
}
```

move함수와 rotate함수는 원리가 같다. 그래서 동작 알고리즘을 위주로 설명하겠다. 우선 핵심적인 동작 알고리즘은 다음에 블록이 이동할 위치를 기록하고 fixedBoard가 true가 아닌 배경들을 BACKGROUND로 모두 채워서 초기화한다. 그 후 이동된 블록의 좌표를 색으로 다시 채워주면 된다. 색을 다시 채워주는 작업을 하기 이전에 isCollied를 통해서 이동한 위치 혹은 회전한 위치에 충돌 발생 여부를 확인한다. 문제가 없다면 색을 다시 채워준다.

rotate()함수에서 newRotation은 다음 회전한 위치의 첫번째 인덱스를 가리킨다. 초기의 rotation값이 0이기 때문에 rotation이 가능한 값은 0, 4, 8, 12 중 하나가 된다. 해당 인덱스를 포함한 다음 4개 인덱스까지가 해당 회전의 블록 좌표들이 된다. 이 원리를 통해서 블록이 회전한다.

2-12. load(), save() 함수

```java
void save() throws IOException {
    String path = "C:\\save";
    String fileName = "save.txt";
    File folder = new File(path);

    if (!folder.exists()) {
        try {
            folder.mkdir();
        }catch (Exception e) {
            e.printStackTrace();
        }
    }

    FileOutputStream output = null;

    try {
        output = new FileOutputStream( name: path+"\\"+fileName);
    } catch (FileNotFoundException e) {
        e.printStackTrace();
    }

    String data = new String();
    for (int i = 0; i < 20; i++) {
        for (int j = 0; j < 10; j++){
            if (fixedBoard[i][j]) {
                data += "1 ";
            }else {
                data += "0 ";
            }
        }
        data += "\n";
    }

    output.write(data.getBytes());
    output.close();
}
```

```java
void load() {
    FileInputStream input = null;

    File file = new File( pathname: "C:\\save\\save.txt");

    try {
        input = new FileInputStream(file);
        int i = 0;
        int row = 0;
        int col = 0;

        while ((i = input.read()) != -1) {
            if ((char)i == ' ' || (char)i == '\n') continue;

            if (col > 9) {
                row++;
                col = 0;
            }
            if ((char)i == '1') fixedBoard[row][col] = true;
            else if ((char)i == '0') fixedBoard[row][col] = false;

              System.out.println(row+"행 "+col+"열 : "+(char)i);
            col++;
        }

    } catch (FileNotFoundException e) {
        e.printStackTrace();
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

우선 save()함수의 경우 fixedBoard변수의 형태를 저장한다. load()함수가 2차원 배열의 형태로 값을 가져오고 저장시키기 때문에 데이터도 해당 형태로 기록했다. fixedBoard가 true면 1, false면 0으로 저장한다. 이는 나중에 한 글자 단위로 데이터를 불러오기 쉽게 하기위해서 이렇게 저장했다.

load()함수는 읽어온 데이터가 1이나 0인 경우에 맞춰서 fixedBoard에 조건에 맞춰 값을 기록해 준다.