**2009190708 이하정**

**소스코드 제출**

PUZZLE GAME CLASS

**package** pros;

**import** java.util.Random;

**import** java.awt.\*;

**import** java.awt.event.\*;

**import** java.awt.image.ImageObserver;

**import** javax.swing.\*;

**public** **class** PuzzleGame **implements** ActionListener, MouseListener, ImageObserver{

JFrame frame;

**public** PuzzleHelper ph = **new** PuzzleHelper();

//출력을 깔끔하기 위한 상수

**final** **int** menu\_size = 0;

//불록의 빈자리를 나타내는 상수

**final** **int** empty = -1;

//블록의 난이도를 조정하는 상수

**final** **int** blk3 = 3;

**final** **int** blk4 = 4;

**final** **int** blk5 = 5;

**final** **int** blk6 = 6;

//게임의 경과시간을 위한 상수

String timerBuffer;

**int** oldTime;

**boolean** start;

//이미지를 불러오기 위한 상수

Image image;

//이미지의 블록을 화면에 출력시키기 위한 2차원 배열, 각각의 배열의 위치를 가지고 있다

**int**[][] Block;

//출력될 스크린의 어디에 그려야 하는지 저장한다

**int**[] Result;

//현재의 난이도를 저장한다

**int** status;

//난이도에 따라 출력될 폭을 정한다

**int** width;

//메뉴들

JMenuBar mb;

JMenu menuselect;

JMenuItem new\_game;

JMenu difficulty;

JMenuItem dan3;

JMenuItem dan4;

JMenuItem dan5;

JMenuItem dan6;

**public** **void** setUpGame(){

//메뉴추가

frame = **new** JFrame();

MyDrawPanel drawPanel = **new** MyDrawPanel();

frame.getContentPane().add(drawPanel);

mb = **new** JMenuBar();

menuselect = **new** JMenu("Menu");

new\_game = **new** JMenuItem("Start new Game");

difficulty = **new** JMenu("Choose Level");

dan3 = **new** JMenuItem("3\*3");

dan4 = **new** JMenuItem("4\*4");

dan5 = **new** JMenuItem("5\*5");

dan6 = **new** JMenuItem("6\*6");

start = **false**;

difficulty.add(dan3);

difficulty.add(dan4);

difficulty.add(dan5);

difficulty.add(dan6);

menuselect.add(new\_game);

menuselect.add(difficulty);

mb.add(menuselect);

frame.setJMenuBar(mb);

//actionlistener를 단다

new\_game.addActionListener(**this**);

dan3.addActionListener(**this**);

dan4.addActionListener(**this**);

dan5.addActionListener(**this**);

dan6.addActionListener(**this**);

image = **new** ImageIcon("picture.jpg").getImage();

//처음 난이도는 3\*3

status = blk3;

width = 300/status;

Block = **new** **int**[36][4];

Result = **new** **int**[36];

//배열을 채워준다

startPlaying();

frame.setSize(300 ,300 + menu\_size);

frame.setVisible(**true**);

frame.addMouseListener(**this**);

frame.repaint();

}//end of setUpGame

**public** **void** startPlaying(){

//시간상태 초기화

start = **false**;

//원본의 블록들의 위치를 한개씩 정해진 난이도에 따라 가져오면서

//각각의 블록의 좌표를 받아온다

**for**(**int** i = 0 ; i < status\*status ; i++){

**for**(**int** j = 0 ; j < 4 ; j++){

**if**(j == 0)

Block[i][j] = (i%status)\*width;

**else** **if**(j == 1)

Block[i][j] = (i/status)\*width;

**else** **if**(j == 2)

Block[i][j] = ((i%status)\*width)+width;

**else**

Block[i][j] = (i/status)\*width+width;

}

}

//결과값 블록 초기화

**for**(**int** k = 0 ; k < status\*status ; k++)

Result[0] = empty;

//랜덤하게 블록을 섞는다

ph.rand\_board(status, Result);

//만약 한개도 안섞일 경우

**while**(ph.end\_check(status, Result) == **true**)

ph.rand\_board(status, Result); //다시 섞는다

}//end of startPlaying

**class** MyDrawPanel **extends** JPanel{

**public** **void** paint(Graphics g) {

//메뉴바를 출력

frame.setJMenuBar(mb);

**super**.paint(g);

**for**(**int** i = 0 ; i < status\*status ; i++) {

//빈블록은 패스

**if**(Result[i] == empty)

**continue**;

//원본블록을 출력위치를 결정하여 그린다

drawbox(Block[Result[i]],Block[i],g);

}

} //end of paint

//좌표4개씩을 가져와서 그림을 출력한다. input이 입력좌표, output이 출력좌표

**public** **void** drawbox(**int**[] input,**int**[] output,Graphics g){

g.drawImage(image,output[0],output[1] + menu\_size,output[2],output[3] + menu\_size,

input[0],input[1],input[2],input[3],**this**);

}//end of drawbox

}//end of MDP

//메뉴처리

**public** **void** actionPerformed(ActionEvent ae){

String name;

name = ae.getActionCommand();

//새로게임을 시작할 경우

**if**(name.equals("새 게임"))

startPlaying();

**else** **if**(name.equals("3\*3")) //난이도에 따라 상수값을 입력

status = blk3;

**else** **if**(name.equals("4\*4"))

status = blk4;

**else** **if**(name.equals("5\*5"))

status = blk5;

**else** **if**(name.equals("6\*6"))

status = blk6;

**else**

**return**;

//폭을 설정하고 블록을 다시 섞는다

width = 300/status;

startPlaying();

//다시 그린다

frame.repaint();//frame add

}

//마우스가 눌렸을때 이벤트처리

**public** **void** mousePressed(MouseEvent e){

**int** hour, min, sec, secs;

**int** x,y;

**boolean** paint = **false**;

x = e.getX();

y = e.getY();

//마우스가 눌린게 처음이면

**if**(start == **false**){

start = **true**;

//시간 기록 시작

oldTime = (**int**) System.*currentTimeMillis*() / 1000;

}

//어디서 눌렸는지 확인한다

**for**(**int** i = 0 ; i < status\*status ; i++){

**if**(ph.CheckPoint(Block[i],x,y)){

//이동체크후 이동

paint = ph.CheckMove(i, status, Result);

**break**;

}

}

//움직일 수 없는 블록을 클릭 했을 경우는 다시 그리지 않는다

**if**(paint == **true**)

frame.repaint();

//게임이 끝나면

**if**(ph.end\_check(status, Result)){

//시간 기록

secs = ((**int**) System.*currentTimeMillis*() / 1000) - oldTime;

//시간을 환산한다

sec = secs % 60;

min = secs / 60 % 60;

hour = secs / 3600;

timerBuffer = String.*format*("%02d:%02d:%02d", hour, min, sec);

//시간을 출력하고 창을 띠운다

JOptionPane.*showMessageDialog*(frame,"Clear Time : " + timerBuffer +

"\nCongratulations!","Clear",

JOptionPane.*INFORMATION\_MESSAGE*);//

startPlaying();

frame.repaint();

}

}

//메인 함수..

**public** **static** **void** main(String args[]){

PuzzleGame pg = **new** PuzzleGame();

pg.setUpGame();

pg.startPlaying();

}//end of main

@Override

**public** **boolean** imageUpdate(Image img, **int** infoflags, **int** x, **int** y,

**int** width, **int** height) {

**return** **false**;

}

**public** **void** mouseClicked(MouseEvent e) { }

**public** **void** mouseReleased(MouseEvent e) { }

**public** **void** mouseEntered(MouseEvent e) { }

**public** **void** mouseExited(MouseEvent e) { }

}//end of PuzzleGame

PUZZLE HELPER CLASS

**package** pros;

**import** pros.PuzzleGame;

**import** java.util.Random;

**import** java.awt.\*;

**import** java.awt.event.\*;

**import** java.awt.image.ImageObserver;

**import** javax.swing.\*;

**public** **class** PuzzleHelper **implements** ActionListener, MouseListener, ImageObserver{

**int** empty = -1;

**public** **boolean** CheckPoint(**int**[] input, **int** x, **int** y)

{

//눌린점이 블록내부에 있는지 검사한다

**if**(input[0] <= x && input[1] <= y

&& x <= input[2] && y <= input[3])

**return** **true**;

**else**

**return** **false**;

}

**public** **boolean** CheckMove(**int** input, **int** status, **int** [] Result)

{

//블록이 맨윗쪽에 있지 않은 경우

**if**(input/status != 0)

{

//윗쪽의 블록이 비어 있는지 검사한다

**if**(Result[input-status] == empty)

{

Result[input-status] = Result[input];

Result[input] = empty;

**return** **true**;

}

}

//블록이 맨오른쪽에 있지 않은 경우

**if**(input%status != status-1)

{

//오른쪽의 블록이 비어 있는지 검사한다

**if**(Result[input+1] == empty)

{

Result[input+1] = Result[input];

Result[input] = empty;

**return** **true**;

}

}

//블록이 맨아랫쪽에 있지 않은 경우

**if**(input/status != status-1)

{

//아랫쪽의 블록이 비어 있는지 검사한다

**if**(Result[input+status] == empty)

{

Result[input+status] = Result[input];

Result[input] = empty;

**return** **true**;

}

}

//블록이 맨왼쪽에 있지 않은 경우

**if**(input%status != 0){

//왼쪽의 블록이 비어 있는지 검사한다

**if**(Result[input-1] == empty){

Result[input-1] = Result[input];

Result[input] = empty;

**return** **true**;

}

}

**return** **false**;

}

//CheckMove설명 : 어떤 임의의 블록이 상하좌우가 다 막혀있는지를 검사하고 막혀있다면, 모든 if문을 들어갈수 없고

//따라서 실행이 될수 없다. 그러나, 그렇지 않다면, 사용자의 입력에 따라 행(+-status)열(+-1)로 움직인다

**public** **void** rand\_board(**int** status, **int**[] Result){

//난수 발생

Random r = **new** Random();

**int** index, x, y;

//결과값 배열에 값을 랜덤하게 채워넣는다

**for**(x = 0 ; x < (status\*status)-1 ; x++){

index = r.nextInt((status\*status)-1);//(status\*status)-1범위 내에서 난수 발생

**for**(y = 0 ; y < x ; y++)

**if**(index == Result[y]) //똑같은 난수가 발생하지 않게 하기 위하여

**break**;

**if**(y == x)

Result[x] = index;

**else**

x--;

}

//마지막 블록은 빈블록이므로 empty로 채운다

Result[x] = empty;

}//end of rand\_board

**public** **boolean** end\_check(**int** status, **int**[] Result) {

//모든 블록이 제자리에 와있는지 확인한다

**for**(**int** i = 0 ; i < (status\*status)-1 ; i++){

**if**(Result[i] != i) //한개라도 틀리면 false리턴

**return** **false**;

}

**return** **true**;

}//end of end\_check

@Override

**public** **boolean** imageUpdate(Image img, **int** infoflags, **int** x, **int** y,

**int** width, **int** height) {

**return** **false**;

}

**public** **void** mouseClicked(MouseEvent e) { }

**public** **void** mousePressed(MouseEvent e) { }

**public** **void** mouseReleased(MouseEvent e) { }

**public** **void** mouseEntered(MouseEvent e) { }

**public** **void** mouseExited(MouseEvent e) { }

**public** **void** actionPerformed(ActionEvent e) { }

}//end of PuzzleHelper