

프로그램 실행 방법

DOS 창에 다음과 같이 입력

Java –jar brain\_attacker2.jar 위로 누적될 때 가중치 , 같은 라인 빈 공간 가중치 , 밑에 빈 공간 가중치 , 라인 지울 때 가중치 , 테스트 빠르기 , 랜덤 시드

예)

Java –jar brain\_attacker2.jar -2 -1 -3 10 s 5928003286129226752

생성자

Init 매소드에 프로그램 실행 파라미터를 받아옴

**public** MyAI(String[] args) {

**super**();

init(args);

};

초기화 매소드

본 메소드에서는 파라미터를 받아들이고 , 각종 변수의 초기화나 각종 블록의 모양 , 가상의 게임 보드를 만든 후 초기화 한다.

**public** **void** init(String[] args)

{

**try**

{

**this**.score\_y = Integer.*parseInt*(args[0]);

**this**.score\_x = Integer.*parseInt*(args[1]);

**this**.blocking\_score = Integer.*parseInt*(args[2]);

**this**.eraseable = Integer.*parseInt*(args[3]);

**this**.mode = args[4];

System.***out***.println("y line score : "+score\_y+" x line score : "+score\_x+" erase score : "+eraseable+" blocking score : "+blocking\_score);

}

**catch**(Exception e)

{

System.***out***.println("Error : "+e.getMessage());

}

**try**

{

**int** i,t;

timer = 0;

last\_block\_y\_point = zero;

/\*

\* 1 ■ ■ ■ ■

\* 0 ■ ■ ■ ■

\* -1 ■ ■ ■ ■

\* -2 ■ ■ ■ ■

\*/

block = **new** **int**[][]

{// Z S I T O L J

{0,0,0,1 , 0,1,0,0 , 0,0,1,0 , 0,0,0,0 , 0,0,0,0 , 0,1,1,0 , 0,0,1,1},

{0,0,1,1 , 0,1,1,0 , 0,0,1,0 , 0,1,1,1 , 0,0,1,1 , 0,0,1,0 , 0,0,1,0},

{0,0,1,0 , 0,0,1,0 , 0,0,1,0 , 0,0,1,0 , 0,0,1,1 , 0,0,1,0 , 0,0,1,0},

{0,0,0,0 , 0,0,0,0 , 0,0,1,0 , 0,0,0,0 , 0,0,0,0 , 0,0,0,0 , 0,0,0,0},

{0,0,0,0 , 0,0,0,0 , 0,0,0,0 , 0,0,1,0 , 0,0,0,0 , 0,0,0,1 , 0,0,0,0},

{0,1,1,0 , 0,0,1,1 , 1,1,1,1 , 0,1,1,0 , 0,0,1,1 , 0,1,1,1 , 0,1,1,1},

{0,0,1,1 , 0,1,1,0 , 0,0,0,0 , 0,0,1,0 , 0,0,1,1 , 0,0,0,0 , 0,0,0,1},

{0,0,0,0 , 0,0,0,0 , 0,0,0,0 , 0,0,0,0 , 0,0,0,0 , 0,0,0,0 , 0,0,0,0},

{0,0,0,1 , 0,1,0,0 , 0,0,1,0 , 0,0,1,0 , 0,0,0,0 , 0,0,1,0 , 0,0,1,0},

{0,0,1,1 , 0,1,1,0 , 0,0,1,0 , 0,1,1,1 , 0,0,1,1 , 0,0,1,0 , 0,0,1,0},

{0,0,1,0 , 0,0,1,0 , 0,0,1,0 , 0,0,0,0 , 0,0,1,1 , 0,0,1,1 , 0,1,1,0},

{0,0,0,0 , 0,0,0,0 , 0,0,1,0 , 0,0,0,0 , 0,0,0,0 , 0,0,0,0 , 0,0,0,0},

{0,0,0,0 , 0,0,0,0 , 0,0,0,0 , 0,0,1,0 , 0,0,0,0 , 0,0,0,0 , 0,1,0,0},

{0,1,1,0 , 0,0,1,1 , 1,1,1,1 , 0,0,1,1 , 0,0,1,1 , 0,1,1,1 , 0,1,1,1},

{0,0,1,1 , 0,1,1,0 , 0,0,0,0 , 0,0,1,0 , 0,0,1,1 , 0,1,0,0 , 0,0,0,0},

{0,0,0,0 , 0,0,0,0 , 0,0,0,0 , 0,0,0,0 , 0,0,0,0 , 0,0,0,0 , 0,0,0,0},

};

play\_ground = **new** **int**[Board.***HEIGHT***][Board.***WIDTH***];

**for**(i=zero;i<Board.***HEIGHT***;i+=1)

{

**for**(t=zero;t<Board.***WIDTH***;t+=1)

{

play\_ground[i][t] = zero;

}

}

}

**catch**(Exception e)

{

System.***out***.println("init Error : "+e.getMessage());

}

}

키보드 입력 메소드

본 메소드에서는 떨어지는 블록이 바닥까지 다 내려왔는지 판단하고 , 게임 보드를 스캔을 하며 , 가장 최적의 점수를 받은 위치와 모양을 계산해내고 , 계산된 결과를 바탕으로 블록을 최적의 위치까지 이동 및 변형 시킨다. 블록 모양을 회전시킬 때 한줄을 내리고 회전을 해야 플레이 보드 오버플로우가 발생되지 않는다 .

@Override

**protected** ControlKeys command() {

ControlKeys key = ControlKeys.***D***;

Shape curPiece = getCurPiece();

timer += 1;//시스템 특성상 블럭생성시 바로 회전이 불가능하다 따라서 모양을 바꿀수 있는 한 줄을 만들어 주고 사용하도록 하자

**if**(block\_crashed(getCurY()))

{

//System.out.println("crashed");

scanner();

calculate(curPiece);

timer = 0;

set\_last\_block\_y\_position(getCurY());

}

**else**

{

set\_last\_block\_y\_position(getCurY());

**if**(timer > 0)

{

**if**(target\_swap > 0)

{

//System.out.println("swap");

key = ControlKeys.***DOWN***;

target\_swap -= 1;

}

**else**

{

//System.out.println("move");

key = ControlKeys.***D***;

**if**(mode.equals("s"))

{

key = ControlKeys.***SPACE***;

}

**if**(target\_x > getCurX())

{

key = ControlKeys.***RIGHT***;

}

**else** **if**(target\_x < getCurX())

{

key = ControlKeys.***LEFT***;

}

}

}

}

**return** key;

}

블록이 다 떨어졌는지 확인 메소드

현재 블록의 y 위치와 마지막 y 위치를 비교해서 마지막 위치 보다 지금 위치가 높으면 다 떨어진 것이 된다.

**public** **boolean** block\_crashed(**int** now\_position)

{

**if**(last\_block\_y\_point < now\_position)

{

//System.out.println("l : "+last\_block\_y\_point+" n : "+now\_position);

**return** **true**;

}

**return** **false**;

}

마지막 떨어지는 블록의 y 위치를 갱신하는 메소드

**public** **void** set\_last\_block\_y\_position(**int** now\_position)

{

**this**.last\_block\_y\_point = now\_position;

}

블록의 모양을 정수로 변환시켜 반환하는 메소드

**int** get\_tetrominoes(Shape shape)

{

**if**(Tetrominoes.***EMPTY*** == shape.getShape())

{

**return** zero;

}

**else** **if**(Tetrominoes.***ZSHAPE*** == shape.getShape())

{

**return** 1;

}

**else** **if**(Tetrominoes.***SSHAPE*** == shape.getShape())

{

**return** 2;

}

**else** **if**(Tetrominoes.***ISHAPE*** == shape.getShape())

{

**return** 3;

}

**else** **if**(Tetrominoes.***TSHAPE*** == shape.getShape())

{

**return** 4;

}

**else** **if**(Tetrominoes.***SQUARE*** == shape.getShape())

{

**return** 5;

}

**else** **if**(Tetrominoes.***LSHAPE*** == shape.getShape())

{

**return** 6;

}

**else** **if**(Tetrominoes.***JSHAPE*** == shape.getShape())

{

**return** 7;

}

**return** -1;

}

현제 플레이 보드의 상태를 스캔하여 play\_ground 배열에 재배치 한다

0 이면 비어있는 것 , 1 이면 블록이 존재하는 것

**public** **void** scanner()

{

**int** i,t;

System.***out***.println("----");

**for**(i=Board.***HEIGHT*** - 1;i>=zero ;i-=1)

{

**for**(t=zero;t<Board.***WIDTH***;t+=1)

{

**try**

{

**if**(Board.*board*[i\*Board.***WIDTH*** + t] != Tetrominoes.***EMPTY***)

{

//System.out.printf("□");

play\_ground[i][t] = 1;

}

**else**

{

//System.out.printf("■");

play\_ground[i][t] = zero;

}

//System.out.println("<"+i+","+t+"> "+Board.board[(i\*Board.WIDTH)+t]);

}

**catch**(Exception e)

{

//System.out.println("Error : <"+i+","+t+">");

}

}

//System.out.println("");

}

}

최대의 가중치를 얻어내 x 좌표와 모양 회전 횟수를 뽑아낸 그러기 위해선 그 블록이 떨어진 위치를 구해야된다 따라서 get\_fallen\_position\_y 메소드에서 현제 모양 , 현재 모양 회전 횟수 , x 좌표를 가져간다

**public** **void** calculate(Shape target)

{

//System.out.println("now shape : "+get\_tetrominoes(target));

**int** i,t,now\_shape = get\_tetrominoes(target),expect\_y , min\_y = Board.***HEIGHT***,max\_score = -0x7fffffff,

save\_score,result\_x = 0,result\_swap = 0;

**for**(i=zero;i<Board.***WIDTH***;i+=1)

{

**for**(t=0;t<swap\_various;t+=1)

{

//다 떨어졌을때의 y 좌표를 구한다

//System.out.println("fall y : "+get\_fallen\_position\_y(now\_shape - 1,t,i)+" x : "+i+" s : "+t);//모양 데이터는 0 부터 시작하므로 -1 , 몇번째 모양 바뀐건지 , 현재 떨어지고 있는 x 좌표

expect\_y = get\_fallen\_position\_y(now\_shape - 1,t,i);

**if**(expect\_y < Board.***HEIGHT*** - 1)

{

min\_y = expect\_y;

save\_score = simulate(i,min\_y,get\_tetrominoes(target)-1, t);

**if**(save\_score > max\_score)

{

max\_score = save\_score;

result\_x = i ;

result\_swap = t;

System.***out***.println("max score updated <"+i+","+min\_y+"> #"+result\_swap+" : "+max\_score);

}

}

}

}

**if**(now\_shape == 3 && (result\_swap == 1 || result\_swap == 3))

{

target\_x = result\_x -1;

}/\*

else if(now\_shape == 7 && result\_swap == 1)

{

target\_x = result\_x + 1;

}\*/

**else**

{

target\_x = result\_x;

}

target\_swap = result\_swap;

}

아래의 메소드 에서는 현제 떨어지고 있는 블록이 어느 y 좌표 까지 떨어질 지 가늠하는 역할을 한다

**public** **int** get\_fallen\_position\_y(**int** shape , **int** swap , **int** x)

{

**int** min\_y = Board.***HEIGHT*** - 1,i,t,q;

**for**(i=Board.***HEIGHT*** - 1; i >= 0; i -=1)

{

**if**(is\_unable(shape, swap, x, i) == **false** && min\_y > i)

{

min\_y = i;

}

**else** **if**(is\_unable(shape, swap, x, i) == **true** && min\_y < Board.***HEIGHT*** - 3)

{

**break**;

}

}

**return** min\_y;

}

Is\_unable메소드를 통해 해당 좌표에 떨어지고 있는 블록이 해당 좌표에 놓일 수 있는지에 대한 여부를 판단한다

**public** **boolean** is\_unable(**int** shape , **int** swap , **int** x , **int** y)// true : unable , false : able

{

**int** i,t,check\_x,check\_y;

//System.out.println("shape : "+shape+" swap : "+swap+" x : "+x+" y : "+y);

//check\_x = x - mid\_x;

//check\_y = y - mid\_y;

**for**(i=swap\*block\_scale;i<(swap+1)\*block\_scale;i+=1)

{

//check\_x = x - mid\_x;

**for**(t=shape\*block\_scale;t<(shape+1)\*block\_scale;t+=1)

{

check\_x = (x + (t%block\_scale - mid\_x));

check\_y = (y - (i%block\_scale - mid\_y));

//System.out.printf("check <"+x+","+y+"> <"+check\_x+","+check\_y+"> <"+(t%block\_scale - mid\_x)+","+(i%block\_scale - mid\_y)+">");

**if**(block[i][t] != 0)

{

**if**(

check\_y < 0 ||

check\_y >= Board.***HEIGHT*** ||

check\_x < 0 ||

check\_x >= Board.***WIDTH***)

{

//System.out.println(" out");

**return** **true**;

}

**else** **if**(play\_ground[check\_y][check\_x] == 1)

{

//System.out.println(" blocked = "+play\_ground[check\_y][check\_x]);

**return** **true**;

}

}

//check\_x += 1;

//System.out.println();

}

//check\_y+=1;

}

**return** **false**;

}

아래 메소드에서는 실질적인 블록이 그 모양 그 회전된 모양으로 구한 위치에 자리 잡았을 때 나오게 되는 가중치를 계산해 낸다

**public** **int** simulate(**int** x, **int** y ,**int** shape, **int** swap)

{

**int**[][] virtual\_board = **new** **int**[Board.***HEIGHT***][Board.***WIDTH***];

**int** i=0,t=0,q=0,itq=0,score = 0,return\_score = -987654321;

**for**(i=0;i<Board.***HEIGHT***;i+=1)

{

**for**(t=0;t<Board.***WIDTH***;t+=1)

{

virtual\_board[i][t] = play\_ground[i][t] ;

}

}

**try**

{

q = swap\*block\_scale;

//System.out.println("draw "+(y-mid\_y)+"~"+(y-mid\_y+block\_scale-1));

**for**(i=y+1;i>=y+1-block\_scale-1 && i >= 0;i-=1)

{

itq = shape\*block\_scale;

**for**(t=x-mid\_x;t<=x-mid\_x+block\_scale-1 && t<Board.***WIDTH***;t+=1)

{

**if**(block[q][itq] != 0 && i >= 0 && t >= 0)

{

virtual\_board[i][t] = block[q][itq] + 1;

}

itq+=1;

}

q+=1;

}

**for**(i=y+1;i>=y+1-block\_scale-1 && i >= 0;i-=1)

{

**for**(t=x-mid\_x;t<=x-mid\_x+block\_scale-1 && t<Board.***WIDTH***;t+=1)

{

**if**(t >= 0)

{

**if**(virtual\_board[i][t] == 2)

{

**for**(q=i-1;q>=0;q-=1)

{

**if**(virtual\_board[q][t] == 0)

{

score = score + (i \* blocking\_score);

}

**else**

{

**break**;

}

}

virtual\_board[i][t] = 1;

}

}

}

}

//System.out.println("blocking score : "+score);

q = swap\*block\_scale;

//System.out.println("draw "+(y-mid\_y)+"~"+(y-mid\_y+block\_scale-1));

**for**(i=y+1;i>=y+1-block\_scale-1 && i >= 0;i-=1)

{

itq = shape\*block\_scale;

**for**(itq=shape\*block\_scale;itq<(shape+1)\*block\_scale;itq+=1)

{

**if**(block[q][itq] != 0)

{

**int** blocked\_num = 0;

**for**(t=0;t<Board.***WIDTH***;t+=1)

{

**if**(virtual\_board[i][t] == 0)

{

blocked\_num += 1;

}

}

**if**(blocked\_num > 0)

{

score = score + (i \* score\_y ) + blocked\_num\*score\_x;

}

**else**

{

//System.out.println("erase!");

score = score + eraseable;

}

**break**;

}

}

q+=1;

}

return\_score = score;

}

**catch**(Exception e)

{

System.***out***.println("Error : "+e.getMessage()+" i : "+i+" t : "+t+" q : "+q+" itq : "+itq);

}

//System.out.println("s <"+x+","+y+"> sh "+shape+" sw "+swap);

show\_virtual\_board(virtual\_board);

System.***out***.println("this test case score : "+score);

**return** return\_score;

}

아래 메소드에서는 가상의 플레이 보드에서 블록이 그 위치에 자리 잡는다면 의 전제 하에 그림을 그리고 이를 보여준다

**public** **void** show\_virtual\_board(**int**[][] target)

{

**int** i,t;

System.***out***.println("---virtual");

**for**(i=Board.***HEIGHT***-1;i>=0;i-=1)

{

**for**(t=0;t<Board.***WIDTH***;t+=1)

{

**if**(target[i][t] == 1)

{

System.***out***.printf("□");

}

**else**

{

System.***out***.printf("■");

}

}

System.***out***.println();

}

}