

# STRUCTURES

## STARTGAME()

INITIAL A BOARD INCLUDING 16 BOX OBJECTS

AT THE BEGINNING

NORMAL LEVEL: BORN 1 BOX

HARD LEVEL: BORN 4 BOXS



```
public void startGame() {
    board = loadImage(loadImage("board.png"));
    normalBoard = loadImage(loadImage("normalBoard.png"));
    hardBoard = loadImage(loadImage("hardBoard.png"));
    normalBoard = loadImage(loadImage("normalBoard.png"));
    hardBoard = loadImage(loadImage("hardBoard.png"));

    Score = 0;
    checked = false;
    checked = false;
    myTiles = new Tile[16]; //16 tiles

    for (int i = 0; i < myTiles.length; i++) { // Initial 16 tiles with their value=0
        myTiles[i] = new Tile(i);
    }

    for (int i = 0; i < 4; i++) { // add 2 tiles at the begin in random position on board
        addTile();
    }

    if (normal) {
        if (hard) {
            for (int i = 0; i < 4; i++) {
                addTile();
            }
        }
    }
}
```

## ADD TILE()

TAKE A RANDOM POSITION IN A LIST OF EMPTY BOX OBJECTS  
TAKE 1 BOX OBJECT -> RANDOMLY SET ITS VALUE IS 2 OR 4

```
private List<Tile> checkSpace() {
    ArrayList<Tile> list = new ArrayList<Tile>(); // add obj Tile in list if its value=0
    for (int i = 0; i < myTiles.length; i++) {
        if (myTiles[i].isEmpty()) {
            list.add(myTiles[i]);
        }
    }
    return list;
}
```

## CHECKSPACE()

ADD ALL OF THE BOXS OBJECTS INTO THE LIST  
IF THAT BOX OBJECT IS EMPTY ( ITS VALUE=0)

```
private void addTile() {
    List<Tile> list = checkSpace(); //list of available space to add new tile

    if (checkSpace().isEmpty()) { // until all tiles on the board have value=0
        int pos = (int) (Math.random() * list.size()) + 0;
        if (normal) {
            if (Math.random() < 0.5) {
                list.get(pos).value = 2;
            } else {
                list.get(pos).value = 4;
            }
        } else {
            double randomDouble = Math.random();
            randomDouble = randomDouble * 2;
            int randomInt = (int) randomDouble;

            if (randomInt == 0) {
                list.get(pos).value = 2;
            } else if (randomInt == 1) {
                list.get(pos).value = 4;
            }
        }
    }
}
```

# MOVE HORIZONTAL

## ARRAY-STORE PREVIOUS STEP

## MOVELEFT()

```
private void moveLeft() {
    steps++;
    boolean checkToAdd = false;
    Tile[] pre = new Tile[16];
    for (int i = 0; i < pre.length; i++) { // Initial 16 tiles with their value=0
        pre[i] = new Tile(i);
    }

    for (int i = 0; i < 4; i++) {
        Tile[] array1 = getLine(i); //get a line in horizontal including 4 elems
        setLine(i, array1, pre); // call by value

        Tile[] array2 = mergeLine(array1); // arrange then merge elems in that array
        setLine(i, array2, myTiles); // call by reference

        if (checkToAdd && compare2Array(array1, array2)) { // only born a new tile when 2 arrays not same
            checkToAdd = true;
        }

        // Compare origin array with merged array
        undo.push(pre);
        if (checkToAdd) {
            addTile();
        }
        if (hard) {
            addTile();
        }
    }

    private boolean compare2Array(Tile[] array1, Tile[] array2) { //check same array in line
        return Arrays.equals(array1, array2);
    }
}
```

```
private void setLine(int index, Tile[] fromarray, Tile[] toarray) { //start copy from 0 in fromarray
    for (int i = 0; i < 4; i++) {
        for (int x = 0; x < 4; x++) {
            toarray[x + { y = 4 } - fromarray[i];
        }
    }
}
```

(CALL BY REFERENCE) POINT CURRENT ARRAY  
TO MERGED ARRAY (4 BOXS IN HOR..)



```
private void setLine(int index, Tile[] fromarray, Tile[] toarray) { //start copy from 0 in fromarray
    for (int i = 0; i < 4; i++) {
        for (int x = 0; x < 4; x++) {
            toarray[x + { y = 4 } - fromarray[i];
        }
    }
}
```

```
private void mergeLine(Tile[] oldline) {
    LinkedList<Tile> list = new LinkedList<Tile>();
    for (int i = 3; i >= 0 && oldline[i].isEmpty(); i--) {
        int num = oldline[i].value;
        Score += num;
        list.addFirst(new Tile(num));
    }
    if (list.size() == 0) {
        return oldline;
    } else {
        while (list.size() != 4) {
            list.addFirst(new Tile(0));
        }
        return list.toArray(new Tile[4]);
    }
}
```

```
private void mergeLine(Tile[] oldline) {
    LinkedList<Tile> list = new LinkedList<Tile>();
    for (int i = 3; i >= 0 && oldline[i].isEmpty(); i--) {
        int num = oldline[i].value;
        Score += num;
        list.addFirst(new Tile(num));
    }
    if (list.size() == 0) {
        return oldline;
    } else {
        while (list.size() != 4) {
            list.addFirst(new Tile(0));
        }
        return list.toArray(new Tile[4]);
    }
}
```

```
private void mergeLine(Tile[] oldline) {
    LinkedList<Tile> list = new LinkedList<Tile>();
    for (int i = 3; i >= 0 && oldline[i].isEmpty(); i--) {
        int num = oldline[i].value;
        Score += num;
        list.addFirst(new Tile(num));
    }
    if (list.size() == 0) {
        return oldline;
    } else {
        while (list.size() != 4) {
            list.addFirst(new Tile(0));
        }
        return list.toArray(new Tile[4]);
    }
}
```

```
private void mergeLine(Tile[] oldline) {
    LinkedList<Tile> list = new LinkedList<Tile>();
    for (int i = 3; i >= 0 && oldline[i].isEmpty(); i--) {
        int num = oldline[i].value;
        Score += num;
        list.addFirst(new Tile(num));
    }
    if (list.size() == 0) {
        return oldline;
    } else {
        while (list.size() != 4) {
            list.addFirst(new Tile(0));
        }
        return list.toArray(new Tile[4]);
    }
}
```

```
private void mergeLine(Tile[] oldline) {
    LinkedList<Tile> list = new LinkedList<Tile>();
    for (int i = 3; i >= 0 && oldline[i].isEmpty(); i--) {
        int num = oldline[i].value;
        Score += num;
        list.addFirst(new Tile(num));
    }
    if (list.size() == 0) {
        return oldline;
    } else {
        while (list.size() != 4) {
            list.addFirst(new Tile(0));
        }
        return list.toArray(new Tile[4]);
    }
}
```

## MOVERIGHT()

```
private void moveRight() {
    steps++;
    Tile[] pre = new Tile[16];
    for (int i = 0; i < pre.length; i++) { // Initial 16 tiles with their value=0
        pre[i] = new Tile(i);
    }

    boolean checkToAdd = false;
    for (int i = 0; i < 4; i++) {
        Tile[] array1 = getLine(i); //get a line in horizontal including 4 elems
        Tile[] array2 = mergeLine(array1); // arrange then merge elems in that array
        //after merged, add the line of array in array myTiles
        setLine(i, array2, myTiles);
        setLine(i, array1, pre);

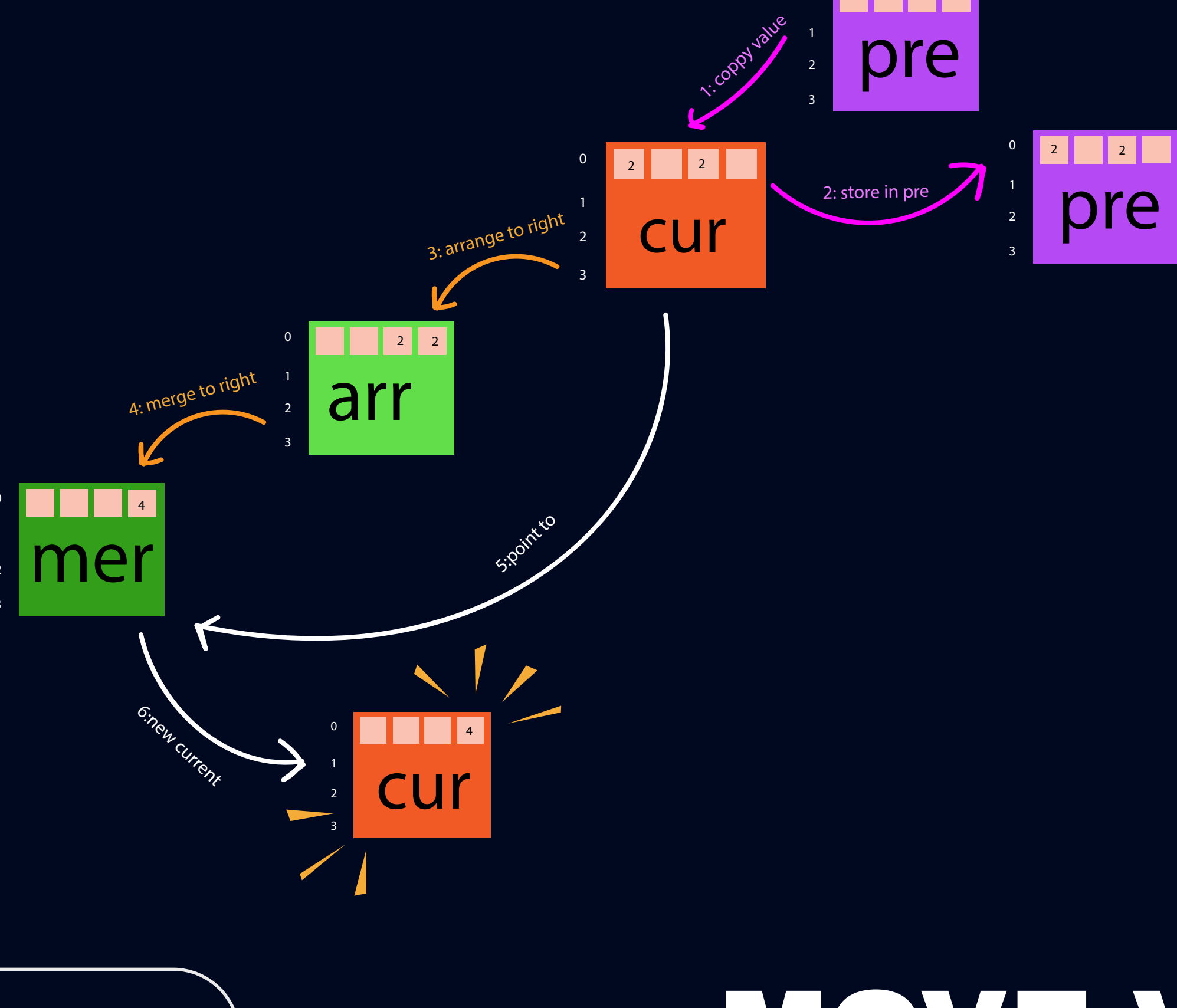
        //after stored 4 lines including 16 elems in each arrays
        if (checkToAdd && compare2Array(array1, array2)) { // only born a new tile when 2 arrays not same
            checkToAdd = true;
        }

        undo.push(pre);

        if (checkToAdd) {
            if (normal) {
                addTile();
            }
            if (hard) {
                addTile();
            }
        }
    }
}
```

```
private void mergeLine(Tile[] oldline) {
    LinkedList<Tile> list = new LinkedList<Tile>();
    for (int i = 3; i >= 0 && oldline[i].isEmpty(); i--) {
        int num = oldline[i].value;
        Score += num;
        list.addFirst(new Tile(num));
    }
    if (list.size() == 0) {
        return oldline;
    } else {
        while (list.size() != 4) {
            list.addFirst(new Tile(0));
        }
        return list.toArray(new Tile[4]);
    }
}
```

```
private void moveLine(Tile[] oldline) {
    LinkedList<Tile> list = new LinkedList<Tile>();
    for (int i = 0; i < 4; i++) {
        if (oldline[i].isEmpty()) {
            list.addFirst(new Tile(0));
        }
    }
    if (list.size() == 0) {
        return oldline;
    } else {
        while (list.size() != 4) {
            list.addFirst(new Tile(0));
        }
        for (int i = 0; i < 4; i++) {
            newLine[i] = list.removeFirst();
        }
        return newLine;
    }
}
```



# MOVE VERTICAL

## MOVEUP()

CURRENT ARRAY  
PREVIOUS ARRAY  
ARRANGED ARRAY  
MERGED ARRAY

```
private void getCol(int index) { //get a col in vertical at 0,4,8,12
    Tile[] col = new Tile[4];
    for (int i = 0; i < 4; i++) {
        col[i] = tileAt(index, i);
    }
    return col;
}
```

```
private void setCol(int index, Tile[] a, Tile[] toarray) { //start copy merged Col array to array myTiles
    for (int i = 0; i < 4; i++) {
        for (int y = 0; y < 4; y++) {
            toarray[x + { y = 4 } - a[i].value;
        }
    }
}
```

```
private void mergeCol(Tile[] oldline) {
    LinkedList<Tile> list = new LinkedList<Tile>();
    for (int i = 0; i < 4 && oldline[i].isEmpty(); i++) {
        int num = oldline[i].value;
        Score += num;
        list.addFirst(new Tile(num));
    }
    if (list.size() == 0) {
        return oldline;
    } else {
        while (list.size() != 4) {
            list.addFirst(new Tile(0));
        }
        return list.toArray(new Tile[4]);
    }
}
```

```
private void mergeCol(Tile[] oldline) {
    LinkedList<Tile> list = new LinkedList<Tile>();
    for (int i = 0; i < 4 && oldline[i].isEmpty(); i++) {
        int num = oldline[i].value;
        Score += num;
        list.addFirst(new Tile(num));
    }
    if (list.size() == 0) {
        return oldline;
    } else {
        while (list.size() != 4) {
            list.addFirst(new Tile(0));
        }
        return list.toArray(new Tile[4]);
    }
}
```

```
private void mergeCol(Tile[] oldline) {
    LinkedList<Tile> list = new LinkedList<Tile>();
    for (int i = 0; i < 4 && oldline[i].isEmpty(); i++) {
        int num = oldline[i].value;
        Score += num;
        list.addFirst(new Tile(num));
    }
    if (list.size() == 0) {
        return oldline;
    } else {
        while (list.size() != 4) {
            list.addFirst(new Tile(0));
        }
        return list.toArray(new Tile[4]);
    }
}
```

```
private void mergeCol(Tile[] oldline) {
    LinkedList<Tile> list = new LinkedList<Tile>();
    for (int i = 0; i < 4 && oldline[i].isEmpty(); i++) {
        int num = oldline[i].value;
        Score += num;
        list.addFirst(new Tile(num));
    }
    if (list.size() == 0) {
        return oldline;
    } else {
        while (list.size() != 4) {
            list.addFirst(new Tile(0));
        }
        return list.toArray(new Tile[4]);
    }
}
```

```
private void mergeCol(Tile[] oldline) {
    LinkedList<Tile> list = new LinkedList<Tile>();
    for (int i = 0; i < 4 && oldline[i].isEmpty(); i++) {
        int num = oldline[i].value;
        Score += num;
        list.addFirst(new Tile(num));
    }
    if (list.size() == 0) {
        return oldline;
    } else {
        while (list.size() != 4) {
            list.addFirst(new Tile(0));
        }
        return list.toArray(new Tile[4]);
    }
}
```

```
private void mergeCol(Tile[] oldline) {
    LinkedList<Tile> list = new LinkedList<Tile>();
    for (int i = 0; i < 4 && oldline[i].isEmpty(); i++) {
        int num = oldline[i].value;
        Score += num;
        list.addFirst(new Tile(num));
    }
    if (list.size() == 0) {
        return oldline;
    } else {
        while (list.size() != 4) {
            list.addFirst(new Tile(0));
        }
        return list.toArray(new Tile[4]);
    }
}
```

```
private void mergeCol(Tile[] oldline) {
    LinkedList<Tile> list = new LinkedList<Tile>();
    for (int i = 0; i < 4 && oldline[i].isEmpty(); i++) {
        int num = oldline[i].value;
        Score += num;
        list.addFirst(new Tile(num));
    }
    if (list.size() == 0) {
        return oldline;
    } else {
        while (list.size() != 4) {
            list.addFirst(new Tile(0));
        }
        return list.toArray(new Tile[4]);
    }
}
```

```
private void mergeCol(Tile[] oldline) {
    LinkedList<Tile> list = new LinkedList<Tile>();
    for (int i = 0; i < 4 && oldline[i].isEmpty(); i++) {
        int num = oldline[i].value;
        Score += num;
        list.addFirst(new Tile(num));
    }
    if (list.size() == 0) {
        return oldline;
    } else {
        while (list.size() != 4) {
            list.addFirst(new Tile(0));
        }
        return list.toArray(new Tile[4]);
    }
}
```

```
private void mergeCol(Tile[] oldline) {
    LinkedList<Tile> list = new LinkedList<Tile>();
    for (int i = 0; i < 4 && oldline[i].isEmpty(); i++) {
        int num = oldline[i].value;
        Score += num;
        list.addFirst(new Tile(num));
    }
    if (list.size() == 0) {
        return oldline;
    } else {
        while (list.size() != 4) {
            list.addFirst(new Tile(0));
        }
        return list.toArray(new Tile[4]);
    }
}
```

```
private void mergeCol(Tile[] oldline) {
    LinkedList<Tile> list = new LinkedList<Tile>();
    for (int i = 0; i < 4 && oldline[i].isEmpty(); i++) {
        int num = oldline[i].value;
        Score += num;
        list.addFirst(new Tile(num));
    }
    if (list.size() == 0) {
        return oldline;
    } else {
        while (list.size() != 4) {
            list.addFirst(new Tile(0));
        }
        return list.toArray(new Tile[4]);
    }
}
```

```
private void mergeCol(Tile[] oldline) {
    LinkedList<Tile> list = new LinkedList<Tile>();
    for (int i = 0; i < 4 && oldline[i].isEmpty(); i++) {
        int num = oldline[i].value;
        Score += num;
        list.addFirst(new Tile(num));
    }
    if (list.size() == 0) {
        return oldline;
    } else {
        while (list.size() != 4) {
            list.addFirst(new Tile(0));
        }
        return list.toArray(new Tile[4]);
    }
}
```

```
private void mergeCol(Tile[] oldline) {
    LinkedList<Tile> list = new LinkedList<Tile>();
    for (int i = 0; i < 4 && oldline[i].isEmpty(); i++) {
        int num = oldline[i].value;
        Score += num;
        list.addFirst(new Tile(num));
    }
    if (list.size() == 0) {
        return oldline;
    } else {
        while (list.size() != 4) {
            list.addFirst(new Tile(0));
        }
        return list.toArray(new Tile[4]);
    }
}
```

```
private void mergeCol(Tile[] oldline) {
    LinkedList<Tile> list = new LinkedList<Tile>();
    for (int i = 0; i < 4 && oldline[i].isEmpty(); i++) {
        int num = oldline[i].value;
        Score += num;
        list.addFirst(new Tile(num));
    }
    if (list.size() == 0) {
        return oldline;
    } else {
        while (list.size() != 4) {
            list.addFirst(new Tile(0));
        }
        return list.toArray(new Tile[4]);
    }
}
```

```
private void mergeCol(Tile[] oldline) {
    LinkedList<Tile> list = new LinkedList<Tile>();
    for (int i = 0; i < 4 && oldline[i].isEmpty(); i++) {
        int num = oldline[i].value;
        Score += num;
        list.addFirst(new Tile(num));
    }
    if (list.size() == 0) {
        return oldline;
    } else {
        while (list.size() != 4) {
            list.addFirst(new Tile(0));
        }
        return list.toArray(new Tile[4]);
    }
}
```

```
private void mergeCol(Tile[] oldline) {
    LinkedList<Tile> list = new LinkedList<Tile>();
    for (int i = 0; i < 4 && oldline[i].isEmpty(); i++) {
        int num = oldline[i].value;
        Score += num;
        list.addFirst(new Tile(num));
    }
    if (list.size() == 0) {
        return oldline;
    } else {
        while (list.size() != 4) {
            list.addFirst(new Tile(0));
        }
        return list.toArray(new Tile[4]);
    }
}
```

```
private void mergeCol(Tile[] oldline) {
    LinkedList<Tile> list = new LinkedList<Tile>();
    for (int i = 0; i < 4 && oldline[i].isEmpty(); i++) {
        int num = oldline[i].value;
        Score += num;
        list.addFirst(new Tile(num));
    }
    if (list.size() == 0) {
        return oldline;
    } else {
        while (list.size() != 4) {
            list.addFirst(new Tile(0));
        }
        return list.toArray(new Tile[4]);
    }
}
```

```
private void mergeCol(Tile[] oldline) {
    LinkedList<Tile> list = new LinkedList<Tile>();
    for (int i = 0; i < 4 && oldline[i].isEmpty(); i++) {
        int num = oldline[i].value;
        Score += num;
        list.addFirst(new Tile(num));
    }
    if (list.size() == 0) {
        return oldline;
    } else {
        while (list.size() != 4) {
            list.addFirst(new Tile(0));
        }
        return list.toArray(new Tile[4]);
    }
}
```

```
private void mergeCol(Tile[] oldline) {
    LinkedList<Tile> list = new LinkedList<Tile>();
    for (int i = 0; i < 4 && oldline[i].isEmpty(); i++) {
        int num = oldline[i].value;
        Score += num;
        list.addFirst(new Tile(num));
    }
    if (list.size() == 0) {
        return oldline;
    } else {
        while (list.size() != 4) {
            list.addFirst(new Tile(0));
        }
        return list.toArray(new Tile[4]);
    }
}
```

```
private void mergeCol(Tile[] oldline) {
    LinkedList<Tile> list = new LinkedList<Tile>();
    for (int i = 0; i < 4 && oldline[i].isEmpty(); i++) {
        int num = oldline[i].value;
        Score += num;
        list.addFirst(new Tile(num));
    }
    if (list.size() == 0) {
        return oldline;
    } else {
        while (list.size() != 4) {
            list.addFirst(new Tile(0));
        }
        return list.toArray(new Tile[4]);
    }
}
```

```
private void mergeCol(Tile[] oldline) {
    LinkedList<Tile> list = new LinkedList<Tile>();
    for (int i = 0; i < 4 && oldline[i].isEmpty(); i++) {
        int num = oldline[i].value;
        Score += num;
        list.addFirst(new Tile(num));
    }
    if (list.size() == 0) {
        return oldline;
    } else {
        while (list.size() != 4) {
            list.addFirst(new Tile(0));
        }
        return list.toArray(new Tile[4]);
    }
}
```

```
private void mergeCol(Tile[] oldline) {
    LinkedList<Tile> list = new LinkedList<Tile>();
    for (int i = 0; i < 4 && oldline[i].isEmpty(); i++) {
        int num = oldline[i].value;
        Score += num;
        list.addFirst(new Tile(num));
    }
    if (list.size() == 0) {
        return oldline;
    } else {
        while (list.size() != 4) {
            list.addFirst(new Tile(0));
        }
        return list.toArray(new Tile[4]);
    }
}
```

```
private void mergeCol(Tile[] oldline) {
    LinkedList<Tile> list = new LinkedList<Tile>();
    for (int i = 0; i < 4 && oldline[i].isEmpty(); i++) {
        int num = oldline[i].value;
        Score += num;
        list.addFirst(new Tile(num));
    }
    if (list.size() == 0) {
        return oldline;
    } else {
        while (list.size() != 4) {
            list.addFirst(new Tile(0));
        }
        return list.toArray(new Tile[4]);
    }
}
```

```
private void mergeCol(Tile[] oldline) {
    LinkedList<Tile> list = new LinkedList<Tile>();
    for (int i = 0; i < 4 && oldline[i].isEmpty(); i++) {
        int num = oldline[i].value;
        Score += num;
        list.addFirst(new Tile(num));
    }
    if (list.size() == 0) {
        return oldline;
    } else {
        while (list.size() != 4) {
            list.addFirst(new Tile(0));
        }
        return list.toArray(new Tile[4]);
    }
}
```

# UNDO | REDO

```
Stack<Tile>[] undo = new Stack<Tile>[10];
Stack<Tile>[] redo = new Stack<Tile>[10];
```

## PRESS UNDO

```
static int undoStep = 0;

public void moveUndo() {
    undoStep++;
    step--;

    if (undoStep == 1) { // If 1st time store that myTiles in RedoStack
        redo.push(myTiles);
    }

    redo.push(undo.peek()); // continue store the top of UndoStack

    myTiles = undo.pop();

    if (undo.size() == 0) {
        System.out.println("cannot Undo anyone ! " + " Step " + step);
        System.out.println("-----");
        undoStep = 0;
    } else {
        System.out.println("Undo " + undoStep + "-- Backward to Step: " + step);
    }
}
```

## PRESS REDO

```
static int redoStep = 0;

public void moveRedo() {
    redoStep++;
    step++;

    if (redoStep == 1) {
        redo.pop();
    }

    undo.push(redo.peek());
    myTiles = redo.pop();

    if (redo.size() == 0) {
        System.out.println("cannot Redo anyone ! " + " Step " + step);
        System.out.println("-----");
        redoStep = 0;
    } else {
        System.out.println("Redo " + redoStep + "-- Forward to Step: " + step);
    }
}
```

WHENEVER YOU MOVE

IF THE 1ST TIME PRESS UNDO

IF THE 1ST TIME PRESS REDO

IF THE 1ST TIME PRESS UNDO

IF THE 1ST TIME PRESS REDO

IF THE 1ST TIME PRESS UNDO

IF THE 1ST TIME PRESS REDO

IF THE 1ST TIME PRESS UNDO

IF THE 1ST TIME PRESS REDO

IF THE 1ST TIME PRESS UNDO

IF THE 1ST TIME PRESS REDO

IF THE 1ST TIME PRESS UNDO

IF THE 1ST TIME PRESS REDO

IF THE 1ST TIME PRESS UNDO

IF THE 1ST TIME PRESS REDO

IF THE 1ST TIME PRESS UNDO

IF THE 1ST TIME PRESS REDO

IF THE 1ST TIME PRESS UNDO

IF THE 1ST TIME PRESS REDO

IF THE 1ST TIME PRESS UNDO

IF THE 1ST TIME PRESS REDO