안드로이드 개발 - 3회

게임으로 배우는 안드로이드 개발

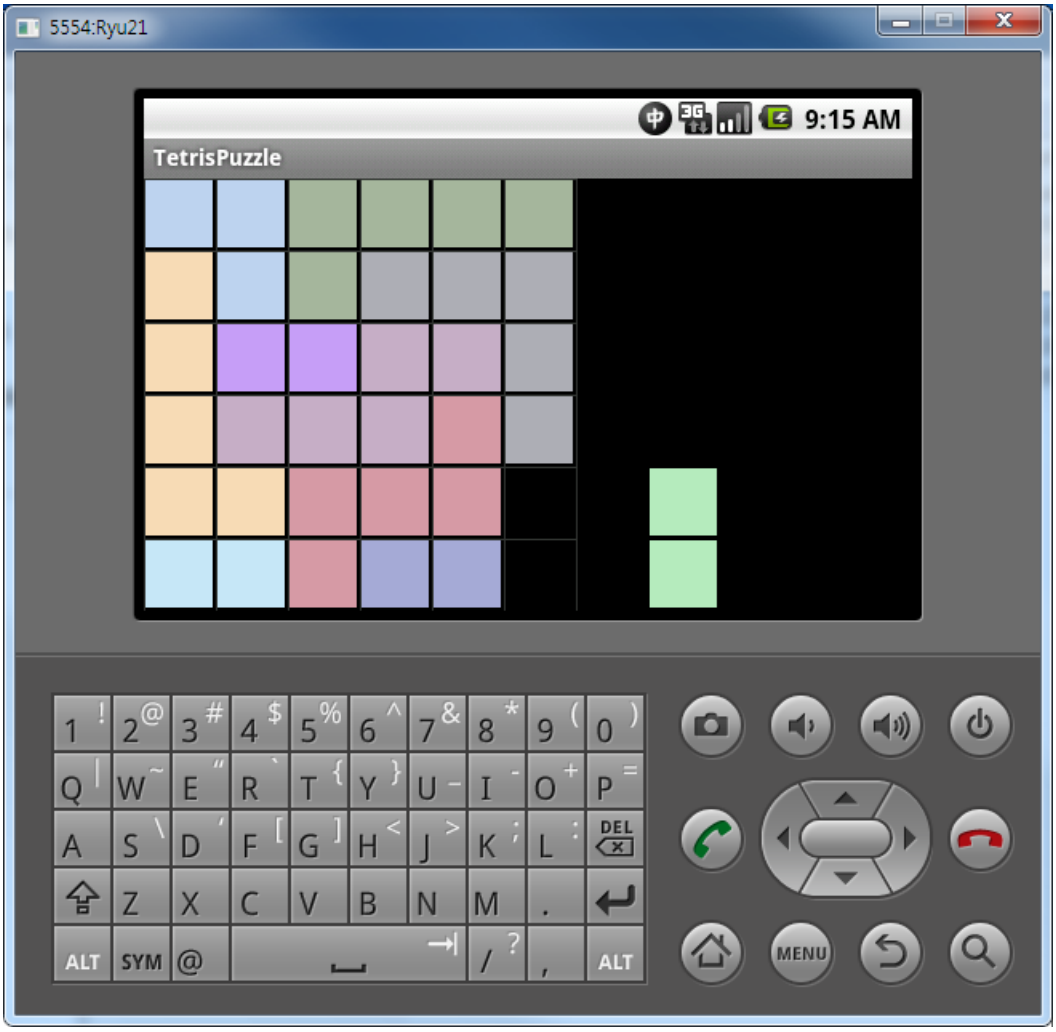
이번 연재에서는 게쪽을 이용하여 테트리스 퍼즐을 만드는 과정을 설명하고자 합니다. 테트리스 블럭 모양의 퍼즐을 모두 맞춰서 사각형을 완성하면 프로그램이 종료되는 게임입니다. 지난 연재에서는 시스템을 객체로 조각 낸 이후에 각 객체끼리의 의존성을 제거하기 위해서 인터페이스를 활용하였습니다. 이번 연재에서는 이벤트 리스너를 통해서 의존성을 제거하는 방식으로 예제를 구성해보았습니다.

글 류종택 ryujt658@hanmail.net
http://ryulib.tistory.com/
(트위터: @RyuJongTaek)

연재순서

1. 심플하고 가벼운 안드로이드 게임 엔진 "게쪽" 소개
2. 슬라이딩 퍼즐 만들기
3. **테트리스 퍼즐 만들기**
4. 슈팅 게임 만들기 #1
5. 슈팅 게임 만들기 #2

1. 테트리스 퍼즐 설계



[그림 1] 테스리스 퍼즐 실행 화면

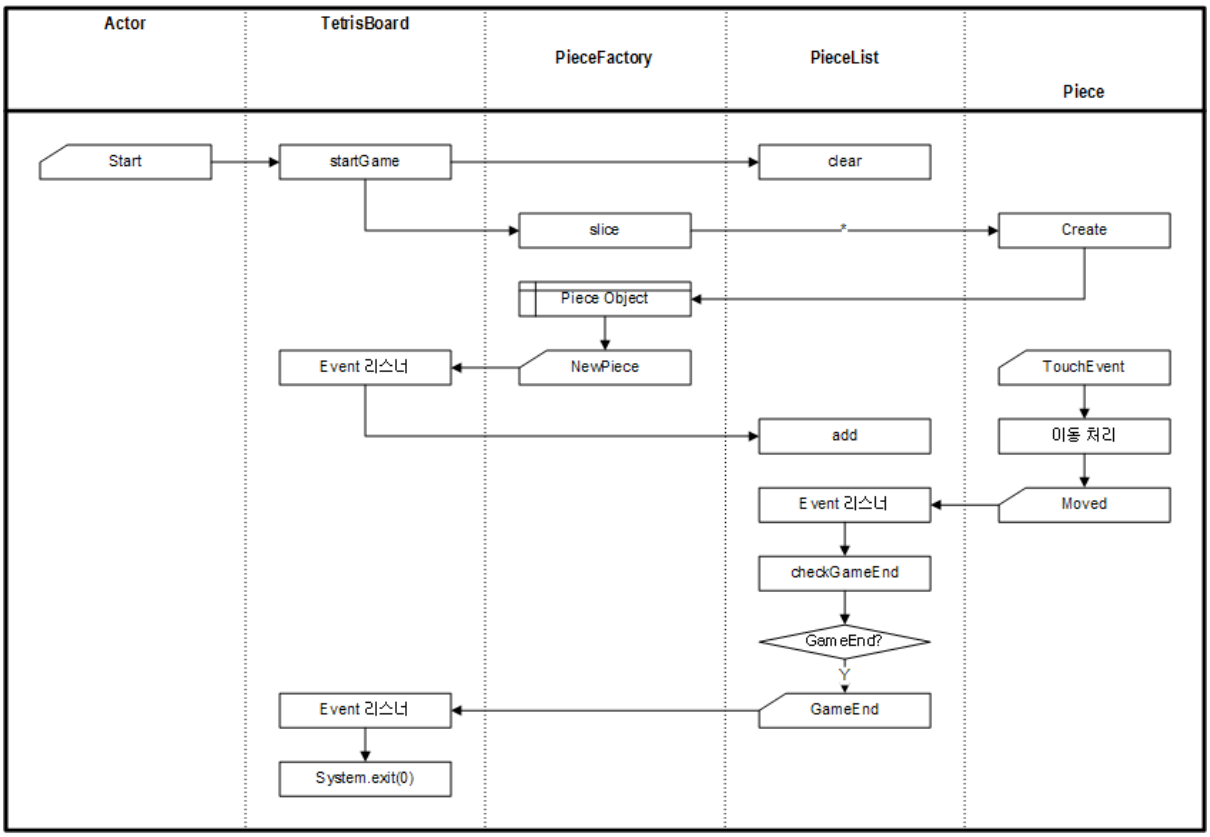
이번에 만들 테트리스 퍼즐은 [그림 1]과 같이 조각난 퍼즐을 다시 하나로 합치는 게임입니다. 이동 중이거나 퍼즐이 겹쳐져 있을 때는 이것을 알 수 있도록 퍼즐 색상을 어둡게(투명) 처리하였습니다.

우선 기능 요구사항을 정리하면 다음과 같습니다.

- 정 사각형의 보드를 무작위로 조각낸다.
- 각 조각들은 터치 이벤트로 이동이 가능해야 한다. 이동이 완료되었을 대는 그리드(정해진 위치)에 맞춰서 정렬되어야 한다.
- 조각이 전부 맞춰지면 어플리케이션을 종료한다.

전반적인 동적 설계는 [그림 2]와 같습니다. 시스템과 완전히 동일한 설명보다는 중요한 부분을 간추려서 설명한 것 입니다.

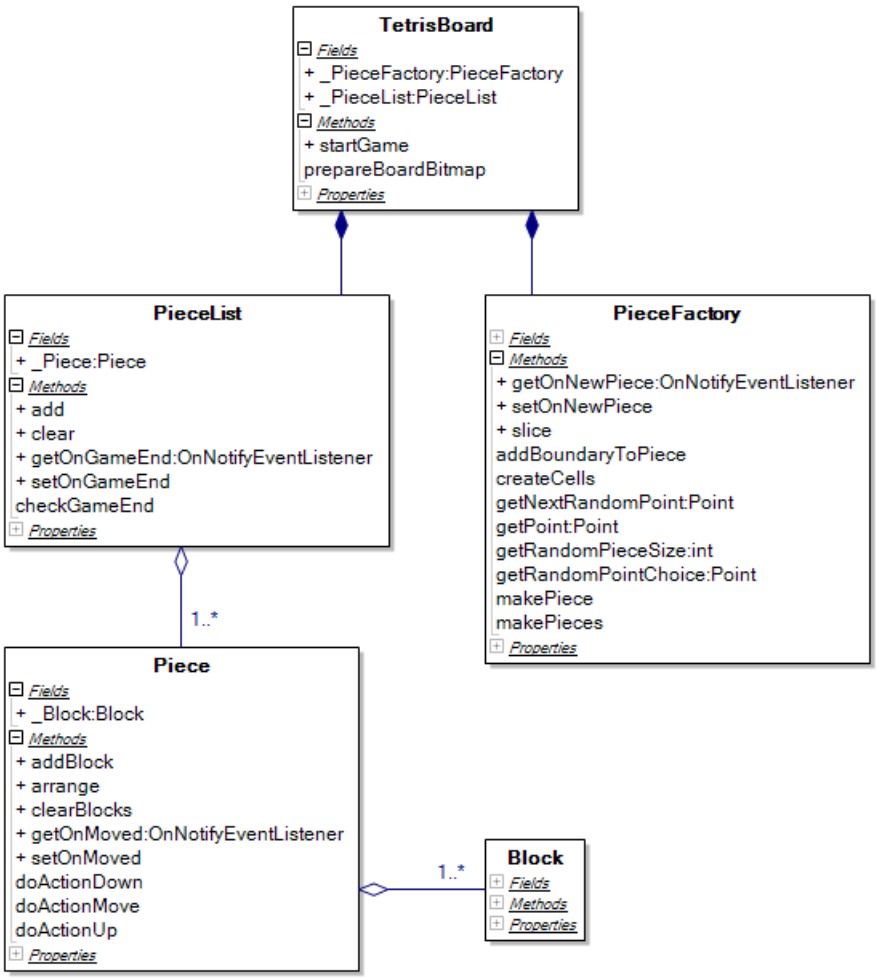
각 모듈들은 서로의 의존성을 이벤트 리스너를 통해서 표현하고 있습니다. 즉, 내가 어떤 이벤트를 발생시킬 것 인지만을 신경 쓰고, 실제 이벤트를 어떻게 핸들링하며 구현할 것인지에 대해서는 신경을 쓰지 않아도 되는 것 입니다.



[그림 2] 게임 소스 전반적인 흐름에 대한 Job Flow

- 게임이 시작되면 TetrisBoard.startGame() 메소드가 실행되고, 이후 전체 조각들이 목록을 관리하는 PieceList.clear()를 통해서 초기화됩니다.
- 바로 이어서 PieceFactory.slice()를 통해서 사각형을 테트리스 블록 모양의 조각(Piece)으로 나눠 줍니다. 이때 랜덤하게 위치를 최대한 서로 겹치지 않도록 배열합니다. slice() 메소드는 새로운 조각을 만들어 내고 배치하고 난 뒤에 OnNewPiece 이벤트를 발생시킵니다.
- OnNewPiece 대한 이벤트 핸들링은 TetrisBoard 에서 진행합니다. TetrisBoard 의 이벤트 리스너는 PieceList.add()를 통해서 생성된 조각을 관리하도록 합니다.
- 조각(Piece)들은 터치 이벤트를 통해서 이동 시킬 수가 있으며, 이동이 완료되면 OnMoved 이벤트를 발생시킵니다.
- OnMoved 이벤트의 핸들링은 상위 객체인 PieceList 의 객체에서 처리합니다. 모든 조각이 제 위치에 있는 지를 점검하고 만약 모든 조각이 제 위치에 있다면, 게임을 종료하는 OnGameEnd 이벤트를 발생시킵니다.
- OnGameEnd 이벤트 역시 상위 객체인 TetrisBoard 에서 처리합니다. 예제에서는 [그림 2]와 같이 System.exit(0)를 통해서 프로그램을 종료시키고 있습니다.

[그림 3]은 동적 분석과 테스트 모듈을 작성하는 동안 완성된, 좀 더 세밀하게 표현된 Class Diagram 입니다. 이것 역시 실제 소스보다는 간략하게 표현되어 있습니다.



[그림 3] Class Diagram

- TetrisBoard 클래스는 PieceList 와 PieceFactory 두 클래스로 구성되어 있습니다.
- PieceList 는 처음의 사각형을 조각 냈을 때 조각들을 관리하게 됩니다. 조각에 해당하는 Piece 는 PieceList 에 복수 개로 연결되어 관리됩니다.
- Piece 는 다시 복수 개의 Block(조그마한 정사각형)으로 이루어져있습니다.
- PieceFactory 는 처음의 사각형을 조각 내는 일을 합니다. 조각 낸 이후에는 최대한 조각들을 서로 떨어지도록 배열합니다.

2. 구현 소스 분석

우선 [소스 1]의 Global 클래스는 지금까지 설명이 되지 않은 것으로 전반적인 흐름에는 크게 관여하지 않기 때문에 설명을 생략했었습니다. Global 에는 프로그램 전반적인 곳에서 공통적으로 사용되는 변수들을 static 으로 선언되어 있습니다. 필자는 복수 개의 객체에서 공통적으로 사용되는 변수들은 static 이나 싱글톤 패턴을 이용하여 처리하고 있습니다.

[소스 1] Global.java

```
 1 : package app.main;
 2 :
 3 : import java.util.Random;
 4 :
 5 : public class Global {
 6 :
 7 : public static void setScreenSize(int width, int height) {
 8 :        screenWidth = width;
 9 :        screenHeight = height;
10 :
11 :        blockSize = screenHeight / boardSize;
12 : }
13 :
14 : public static int screenWidth = 480;
15 :
16 : public static int screenHeight = 320;
17 :
18 : public static int blockSize = 24;
19 :
20 : // BoardSize*BoardSize 크기의 배열
21 : public static int boardSize  = 6;
22 :
23 : private static final Random _Random = new Random();
24 :
25 : public static int getRandom(int n) {
26 :        return _Random.nextInt(n);
27 : }
28 :
29 : }
```

[소스 1]에 지정된 수치들은 실제로는 의미가 없습니다. 프로그램이 시작되면, 7: 라인의 setScreenSize() 메소드가 실행되고, 이후 화면 크기에 맞춰서 조절됩니다. 따라서, 해상도가 조금 상이한 스마트 폰에서 실행된다고 해도 큰 문제 없이 사용될 수 있습니다.

- `screenWidth, screenHeight`: 스마트 폰 화면의 크기 (실제로는 게임 엔진 화면의 크기)
- `blockSize`: 조각들을 이루는 작은 정사각형의 변의 크기
- `boardSize`: 처음의 사각형이 `[boardSize][boardSize]` 크기의 배열로 조각이 납니다.
- `getRandom()`: 랜덤을 사용하기 위해서 객체를 생성하는 번거로움을 피하기 위해서 미리 만들어 두었습니다.

[소스 2] Main.java

```
 1 : package app.main;
```

```
  2 :
  3 : import ryulib.game.GamePlatform;
  4 : import android.app.Activity;
  5 : import android.os.Bundle;
  6 : import android.view.ViewGroup;
  7 : import android.widget.LinearLayout;
  8 :
  9 : public class Main extends Activity {
 10 :
 11 :    /** Called when the activity is first created. */
 12 :    @Override
 13 :    public void onCreate(Bundle savedInstanceState) {
 14 :        super.onCreate(savedInstanceState);
 15 :
 16 :        _GamePlatform = new GamePlatform(this);
 17 :        _GamePlatform.setUseMotionEvent(true);
 18 :        _GamePlatform.setLayoutParams(
 19 :               new LinearLayout.LayoutParams(
 20 :                           ViewGroup.LayoutParams.FILL_PARENT,
 21 :                           ViewGroup.LayoutParams.FILL_PARENT,
 22 :                           0.0F
 23 :               )
 24 :        );
 25 :        setContentView(_GamePlatform);
 26 :
 27    :                                          _TetrisBoard    =    new
TetrisBoard(_GamePlatform.getGameControlGroup());
 28 :    }
 29 :
 30 : private GamePlatform _GamePlatform = null;
 31 : private TetrisBoard _TetrisBoard = null;
 32 :
 33 : }
```

[소스 2]는 Main Activity 의 소스이며, 지금까지 만들어 온 것과 동일하여 특별하게 설명을 드릴 만한 곳은 없기에 설명을 생략합니다.

[소스 3] TetrisBoard.java

```
  1 : package app.main;
                                  .. .. ..
 12 : public class TetrisBoard extends GameControl {
 13 :
 14 : public TetrisBoard(GameControlGroup gameControlGroup) {
 15 :        super(gameControlGroup);
 16 :
 17 :        gameControlGroup.addControl(this);
 18 :
 19 :        _PieceFactory = new PieceFactory(gameControlGroup);
 20 :        _PieceFactory.setOnNewPiece(_OnNewPiece);
 21 :
 22 :        _PieceList.setOnGameEnd(_OnGameEnd);
 23 : }
 24 :
 25 : private PieceFactory _PieceFactory = null;
 26 : private PieceList  PieceList = new PieceList();
 27 :
 28 : private      OnNotifyEventListener       _OnNewPiece      =      new
```

```
OnNotifyEventListener() {
  29 :        @Override
  30 :        public void onNotify(Object sender) {
  31 :               getGameControlGroup().addControl((Piece) sender);
  32 :               _PieceList.add((Piece) sender);
  33 :        }
  34 : };
  35 :
  36 : private       OnNotifyEventListener       _OnGameEnd       =       new
OnNotifyEventListener() {
  37 :        @Override
  38 :        public void onNotify(Object sender) {
  39 :               System.exit(0);
  40 :        }
  41 : };
  42 :
  43 : private    Bitmap    _BoardBitmap    =    Bitmap.createBitmap(1,    1,
Config.ARGB_8888);
  44 : private Canvas _BoardCanvas = new Canvas();
  45 :
  46 : private Canvas _Canvas = null;
  47 : private Paint _Paint = new Paint();
  48 :
  49 : public void startGame() {
  50 :         PieceList.clear();
  51 :        _PieceFactory.slice();
  52 : }
  53 :
  54 : @Override
  55 : protected void onStart(GamePlatformInfo platformInfo) {
  56 :        _Canvas = platformInfo.getCanvas();
  57 :
  58   :                              Global.setScreenSize(_Canvas.getWidth(),
_Canvas.getHeight());
  59 :        setBoardSize(Global.blockSize * Global.boardSize);
  60 :
  61 :        startGame();
  62 : }
  63 :
  64 : @Override
  65 : protected void onDraw(GamePlatformInfo platformInfo) {
  66 :        _Paint.setARGB(255, 0, 0, 0);
  67 :        _Canvas.drawRect(platformInfo.getRect(), _Paint);
  68 :
  69 :        _Canvas.drawBitmap(_BoardBitmap, 0, 0, _Paint);
  70 : }
  71 :
  72 : private int _BoardSize = 0;
  73 :
  74 : public int getBoardSize() {
  75 :        return _BoardSize;
  76 : }
  77 :
  78 : public void setBoardSize(int _BoardSize) {
  79 :        this._BoardSize = _BoardSize;
  80 :        prepareBoardBitmap(_BoardSize);
  81 : }
  82 :
  83 : private void prepareBoardBitmap(int boardSize) {
                                    .. .. ..
```

```
 96 : }
 97 :
 98 : }
```

소스가 길어서 몇 군데의 소스를 생략하였습니다.

19-20: PieceFactory 객체를 생성하고, OnNewPiece 이벤트를 처리할 리스너를 지정합니다.

PieceList 는 26: 라인에서 객체를 생성하고, 22: 라인에서 OnGameEnd 이벤트를 처리할 리스너를 지정합니다.

28-34: PieceFactory 가 처음의 사각형을 조각 내어 조각 객체를 생성할 때마다 발생하는 OnNewPiece 이벤트를 처리할 리스너를 구현하고 있습니다.

36-41: Piece(조각)들이 이동할 때마다 모든 조각들이 제 위치에 있는 지를 확인하고, 모두 제 위치에 있을 경우에 발생하는 OnGameEnd 이벤트를 처리할 리스너 입니다. 39: 라인을 통해서 모든 조각들이 제 위치에 있을 경우에는 프로그램을 종료 합니다.

83-96: 배경에 쓰일 Bitmap 이미지를 작성합니다. Global.boardSize 에 맞도록 바둑판 모양의 격자를 그리는 부분이며 소스는 생략하였습니다. 매번 바둑판을 그리는 것이 비효율적이기에 Bitmap 이미지를 준비하여 재사용하도록 하였습니다. 이미지를 미리 준비하여 리소스로 지정하여 불러 들여도 됩니다.

58: 프로그램이 시작되자 마자 Global.setScreenSize() 메소드를 호출하여 게임 엔진의 화면 크기를 지정합니다. 이것으로 필요한 모든 변수들이 초기화 됩니다.

61: 게임을 시작합니다. startGame()가 58: 라인의 Global.setScreenSize()보다 나중에 실행되어야 하는 것을 유의하시기 바랍니다.

49-52: [그림 2]에서 본 것과 같이 게임이 시작되면, PieceList 를 초기화 하고, PieceFactory 를 통해서 처음의 사각형을 조각 냅니다.

[소스 4] PieceList.java

```
  1 : package app.main;
  2 :
  3 : import java.util.ArrayList;
  4 :
  5 : import ryulib.OnNotifyEventListener;
  6 :
  7 : public class PieceList {
  8 :
  9 : private ArrayList<Piece> _List = new ArrayList<Piece>();
 10 :
 11 : public void clear() {
```

```
 12 :        _List.clear();
 13 : }
 14 :
 15 : private      OnNotifyEventListener      _OnPieceMoved      =      new
OnNotifyEventListener() {
 16 :        @Override
 17 :        public void onNotify(Object sender) {
 18 :              checkGameEnd();
 19 :        }
 20 : };
 21 :
 22 : public void add(Piece piece) {
 23 :        _List.add(piece);
 24 :        piece.setOnMoved(_OnPieceMoved);
 25 : }
 26 :
 27 : private void checkGameEnd() {
                                    .. .. ..
 54 :        if (_OnGameEnd != null) _OnGameEnd.onNotify(this);
 55 : }
 56 :
 57 : private OnNotifyEventListener _OnGameEnd = null;
 58 :
 59 : public void setOnGameEnd(OnNotifyEventListener _OnGameEnd) {
 60 :        this. OnGameEnd =  OnGameEnd;
 61 : }
 62 :
 63 : public OnNotifyEventListener getOnGameEnd() {
 64 :        return _OnGameEnd;
 65 : }
 66 :
 67 : }
```

22-25: PieceFactory에 의해서 만들어진 조각들을 _List 목록에 포함시키면서 이벤트 리스너를 지정하고 있습니다. 이제 각 Piece(조각)들이 움직일 때마다 18: 라인이 실행되어, 소스가 생략된 27-55: 라인에서 모든 조각들이 제 위치에 있는 지를 확인하여, 54: 라인을 통해 리스너를 호출하게 됩니다. 이벤트가 실제 어떻게 처리하는 지는 이벤트 리스너를 보유한 객체에게 위임하고 자신을 신경 쓰지 않습니다.

[소스 5-1] PieceFactory.java #1

```
  1 : package app.main;
                                    .. .. ..
 11 : public class PieceFactory extends GameControl {
 12 :
 13 : public PieceFactory(GameControlGroup gameControlGroup) {
 14 :        super(gameControlGroup);
 15 :
 16 : }
 17 :
 18 : private Boundary[][] _Boundaries = null;
 19 : private int _BoundriesCount;
 20 :
 21 : public void slice() {
 22 :        createCells();
 23 :        makePieces();
 24 : }
```

```
 25 :
 26 : private void makePieces() {
 27 :        _BoundriesCount = Global.boardSize * Global.boardSize;
 28 :        while (_BoundriesCount > 0) {
 29 :               makePiece();
 30 :        }
 31 : }
 32 :
 33 : private void makePiece() {
 34 :        int pieceSize  = getRandomPieceSize();
 35 :        Piece piece = new Piece(getGameControlGroup());
 36 :        Point point = getRandomPointChoice();
 37 :
 38 :        addBoundaryToPiece(point, piece);
 39 :        pieceSize--;
 40 :        _BoundriesCount--;
 41 :
 42 :        while ((pieceSize > 0) && (_BoundriesCount > 0)) {
 43 :               point = getNextRandomPoint(point);
 44 :               if (point == null) break;
 45 :
 46 :               addBoundaryToPiece(point, piece);
 47 :               pieceSize--;
 48 :               _BoundriesCount--;
 49 :        }
 50 :
 51 :        piece.arrange();
                                .. .. ..
 63 :        if (_OnNewPiece != null) _OnNewPiece.onNotify(piece);
 64 : }
 65 :
 66 : private void createCells() {
 67 :        _Boundaries                          =                          new
Boundary[Global.boardSize][Global.boardSize];
 68 :
 69 :        for (int y=0; y<Global.boardSize; y++) {
 70 :               for (int x=0; x<Global.boardSize; x++) {
 71 :                      _Boundaries[x][y]              =              new
Boundary(x*Global.blockSize,   y*Global.blockSize,   (x+1)*Global.blockSize,
(y+1)*Global.blockSize);
 72 :               }
 73 :        }
 74 : }
```

소스가 너무 길어서 나누어서 설명하고자 합니다.

21-24: PieceFactory의 가장 중요한 역할인 조각내기에 해당 합니다.

66-74: 우선 조각을 내기 전에 Global.boardSize의 2차원 배열로 Boundary 객체를 생성합니다. 즉, 최초의 사각형을 boardSize *boardSize 개수만큼의 정사각형으로 조각 냅니다.

26-31: 정사각형의 조각의 개수만큼 반복하면서 이것들을 서로 랜덤한 모양과 랜덤한 개수로 묶어 줍니다. 이것이 마치 테트리스 블록처럼 보이게 됩니다. 그리고, 이러한 묶음을 Piece라고 부르겠습니다.

33-64: 실제 조각을 Piece로 묶어주는 곳 입니다. 63: 라인에서 묶어진 Piece를 이벤트를 발생시켜 리스너에게 알려주고 있습니다.

[소스 5-2] PieceFactory.java #2

```
 76 : private void addBoundaryToPiece(Point point, Piece piece) {
 77 :       _Boundaries[point.x][point.y] = null;
 78 :       piece.addBlock(point.x, point.y);
 79 : }
 80 :
 81 : private Point getRandomPointChoice() {
 82 :       int x = Global.getRandom(Global.boardSize);
 83 :       int y = Global.getRandom(Global.boardSize);
 84 :
 85 :       while (_Boundaries[x][y] == null) {
 86 :             x = Global.getRandom(Global.boardSize);
 87 :             y = Global.getRandom(Global.boardSize);
 88 :       }
 89 :
 90 :       return new Point(x, y);
 91 : }
 92 :
 93 : // 해당 좌표에 Boundary를 가져 올 수 있는 가?  있으면 좌표를 리턴한
 94 : private Point getPoint(int x, int y) {
 95 :       if ((x < 0) || (x >= Global.boardSize)) return null;
 96 :       if ((y < 0) || (y >= Global.boardSize)) return null;
 97 :
 98 :       if (_Boundaries[x][y] != null) {
 99 :             return new Point(x, y);
100 :             } else {
101 :                   return null;
102 :             }
103 :       }
104 :
105 :       private Point getNextRandomPoint(Point basePoint) {
106 :             ArrayList<Point> points = new ArrayList<Point>();
107 :
108 :             Point point;
109 :
110 :             point = getPoint(basePoint.x-1, basePoint.y);
111 :             if (point != null) points.add(point);
112 :
113 :             point = getPoint(basePoint.x+1, basePoint.y);
114 :             if (point != null) points.add(point);
115 :
116 :             point = getPoint(basePoint.x, basePoint.y-1);
117 :             if (point != null) points.add(point);
118 :
119 :             point = getPoint(basePoint.x, basePoint.y+1);
120 :             if (point != null) points.add(point);
121 :
122 :             if (points.size() == 0) {
123 :                   return null;
124 :             } else {
125 :                   int index = Global.getRandom(points.size());
126 :                   return points.get(index);
127 :             }
```

```
 128 :        }
 129 :
 130 :        private int getRandomPieceSize() {
 131 :               // TODO Auto-generated method stub
 132 :               return 5;
 133 :        }
 134 :
 135 :        public  void  setOnNewPiece(OnNotifyEventListener  _OnNewPiece)
{
 136 :               this._OnNewPiece = _OnNewPiece;
 137 :        }
 138 :
 139 :        public OnNotifyEventListener getOnNewPiece() {
 140 :               return _OnNewPiece;
 141 :        }
 142 :
 143 :        private OnNotifyEventListener _OnNewPiece = null;
 144 :
 145 : }
```

76-78: Piece는 이미 설명한 것과 같이 복수 개의 Block으로 구성되어 있습니다. 77: 라인에서는 Piece로 구성될 위치의 정사각형이 이미 사용되었음을 선언합니다. null로 표시하여 더 이상 사용할 수 없음을 알리고 있습니다. 78: 라인에서는 해당 블록을 Piece에 구성시킵니다.

81-91: _Boundaries에서 아무 정사각형(Block)이나 선택합니다. null로 지정되지 않은 넘만을 선별합니다. 36: 라인에서처럼 처음에는 아무 것이나 랜덤하게 선택하여 Piece로 구성합니다. 이후에는 근처에 있는 정사각형을 찾아서 Piece에 포함시킵니다. 대각선 방향이나, 동 떨어져 있는 것을 하나로 묶어둘 수는 없기 때문입니다. 이때, 105-128: 에서는 전후/좌우에 있는 정사각형 중 null이 아닌 하나를 선별하여 Piece에 묶어줍니다.

130-133: 이 부분은 34: 라인에서 호출되며, Piece에 묶여질 수 있는 정사각형의 최대 개수를 나타냅니다. 5를 리턴 하는 것으로 고정되어 있지만, 추후 게임의 재미를 위해서 랜덤하게 또는 레벨에 맞춰서 변동시킬 부분 입니다.

[소스 6-1] Piece.java #1

```
  1 : package app.main;
                                  .. .. ..
 14 : public class Piece extends GameControl {
 15 :
 16 : public Piece(GameControlGroup gameControlGroup) {
 17 :        super(gameControlGroup);
 18 :
 19 :        gameControlGroup.addControl(this);
 20 : }
 21 :
 22 : private HitArea _HitArea = new HitArea();
 23 :
 24 : @Override
 25 : protected HitArea getHitArea() {
 26 :        return _HitArea;
 27 : }
```

19: 모든 GameControl은 GameControlGroup에 addControl() 메소드로 등록되어야 화면에 표시되는 등에 동작이 이루어지는 것에 유의하시기 바랍니다.

이 소스에서는 상당히 중요한 충돌 처리 부분이 존재합니다. 우선 22: 라인처럼 HitArea 객체가 필요합니다. 이것은 자신의 영역을 알리는 역할을 합니다. 또한, 이 객체는 복수 개의 Boundary를 묶어서 사용할 수 있도록 되어 있습니다. 이것은 비행기와 달리 테트리스 블록은 모양이 복잡하기 때문입니다. 처리 속도를 위해서 곡선대신 네모 모양의 Boundary를 여러 개 묶어서 사용하고 있습니다. 24-27: 라인에서는 GameControlBase에 선언되어 있는 getHitArea() 메소드를 재정의하고 있습니다. 여기서 우리가 생성한 HitArea 객체를 되돌려 주기만 하면, 게임 엔진이 충돌 체트를 할 때, 방금 넘겨준 HitArea 객체를 통해서 나의 위치를 파악하고 다른 객체들과 충돌했는 지 여부를 알려주게 됩니다.

실제 충돌되는 위치를 지정하는 곳은 191-194: 입니다. HitArea 객체에 현재 Piece가 가지고 있는 Block()들의 Boundary 객체를 묶어 줍니다. 이것을 한 번에 끝내지 않고 매번 Piece가 움직일 때마다 다시 계산하는 이유는, Piece가 움직일 때마다 HitArea도 같이 움직일 수 밖에 없기 때문입니다.

[소스 6-2] Piece.java #2

```
 29 : private int _X = 0;
 30 : private int _Y = 0;
 31 : private int _MinLeft = 0xFFFF;
 32 : private int _MinTop = 0xFFFF;
 33 : private int _MaxLeft = -1;
 34 : private int _MaxTop = -1;
 35 :
 36 : ArrayList<Block> _Blocks = new ArrayList<Block>();
 37 :
 38 : public int getWidth() {
 39 :        return _MaxLeft - _MinLeft + 1;
 40 : }
 41 :
 42 : public int getHeight() {
 43 :        return _MaxTop - _MinTop + 1;
 44 : }
 45 :
 46 : public void clearBlocks() {
 47 :        _MinLeft = 0xFFFF;
 48 :        _MinTop = 0xFFFF;
 49 :
 50 :        _Blocks.clear();
 51 : }
 52 :
 53 : public void addBlock(int x, int y) {
 54 :        if (x < _MinLeft) _MinLeft = x;
 55 :        if (y < _MinTop ) _MinTop  = y;
 56 :
 57 :        if (x > _MaxLeft) _MaxLeft = x;
 58 :        if (y > _MaxTop ) _MaxTop  = y;
 59 :
```

```
 60 :        Block block = new Block(x, y);
 61 :        _Blocks.add(block);
 62 : }
 63 :
 64 : public void arrange() {
 65 :        for (Block block : _Blocks) {
 66 :               block.decX(_MinLeft);
 67 :               block.decY(_MinTop);
 68 :        }
 69 :
 70 :        afterMoved();
 71 : }
 72 :
 73 : private int _A = 255;
 74 : private int _R = (int) (Math.random() * 100) + 155;
 75 : private int _G = (int) (Math.random() * 100) + 155;
 76 : private int _B = (int) (Math.random() * 100) + 155;
 77 :
 78 : @Override
 79 : protected void onDraw(GamePlatformInfo platformInfo) {
 80 :        Paint paint  = platformInfo.getPaint();
 81 :
 82 :        if (_isMoving) {
 83 :               paint.setARGB(100, _R, _G, _B);
 84 :        } else {
 85 :               if ((checkCollision(this) != null)) {
 86 :                      paint.setARGB(100, _R, _G, _B);
 87 :               } else {
 88 :                      paint.setARGB(_A, _R, _G, _B);
 89 :               }
 90 :        }
 91 :
 92 :        int left = _X*Global.blockSize + _TouchMove.x-_TouchDown.x;
 93 :        int top  = _Y*Global.blockSize + _TouchMove.y-_TouchDown.y;
 94 :
 95 :        for (Block block : _Blocks) {
 96 :               platformInfo.getCanvas().drawRect(
 97 :                             block.getBoundary().getRect(left, top),
 98 :                             paint
 99 :               );
100 :               }
101 :        }
```

29-30: Piece의 현재 위치를 나타냅니다. 픽셀 단위가 아니고 화면을 블록 크기로 나누어진 바둑판으로 보았을 때의 좌표입니다.

53-62: PieceFactory를 통해서 묶어서 사용할 Block의 좌표를 입력받으면, 실제 Block 객체를 생성해서 묶어주는 역할을 담당합니다. 이때, MinLeft와 MinTop은 묶어진 블록들 중에서 좌상단에 위치한 불록의 좌표를 계산하기 위해서 입니다. 이후 64-71: 라인의 arrange() 메소드를 통해서 좌상단의 블록의 좌표가 (0, 0)이 되도록 모든 블록들의 좌표를 줄여 나갑니다.

73-76: Piece의 색상을 랜덤하게 결정합니다.

78-101: Piece를 상황에 맞도록 그립니다. 82: 라인에서는 움직이는 도중인가를 파악하고, 85: 라

인에서는 다른 Piece들과 충돌하는 가를 파악합니다. 두 가지에 해당하면, Piece에 투명도를 주어서 상황을 알 수 있도록 합니다.

95-100: Piece에 포함된 블록 전체를 그립니다.

[소스 6-3] Piece.java #3

```
 103 :        // Action Down, Move 일 때, event 발생 위치
 104 :        private Point _TouchDown = new Point();
 105 :        private Point _TouchMove = new Point();
 106 :
 107 :    private void doActionDown(int x, int y) {
 108 :        _TouchDown.set(x, y);
 109 :        _TouchMove.set(x, y);
 110 :    }
 111 :
 112 :    private void doActionUp(int x, int y) {
 113 :        int  cx  =  (_X*Global.blockSize)  +  (x  -  _TouchDown.x)  +
(Global.blockSize / 2);
 114 :        int  cy  =  (_Y*Global.blockSize)  +  (y  -  _TouchDown.y)  +
(Global.blockSize / 2);
 115 :
 116 :        _TouchDown.set(0, 0);
 117 :        _TouchMove.set(0, 0);
 118 :
 119 :        if (cx < 0) cx = 0;
 120 :        if (cy < 0) cy = 0;
 121 :
 122 :        if  (cx  >  (Global.screenWidth    -  getWidth()))    cx  =
Global.screenWidth  - getWidth();
 123 :        if  (cy  >  (Global.screenHeight  -  getHeight()))  cy  =
Global.screenHeight - getHeight();
 124 :
 125 :        cx = (cx / Global.blockSize);
 126 :        cy = (cy / Global.blockSize);
 127 :
 128 :        setPoint(cx, cy);
 129 :    }
 130 :
 131 :        private void doActionMove(int x, int y) {
 132 :              _TouchMove.set(x, y);
 133 :    }
 134 :
 135 :        private boolean _isMoving = false;
 136 :
 137 :    @Override
 138 :      protected boolean onTouchEvent(GamePlatformInfo platformInfo,
MotionEvent event) {
                                    .. .. ..
 171 :    }
 172 :
 173 :        private boolean getIsMyArea(int x, int y) {
 174 :              for (Block block : _Blocks) {
 175 :                    if  (block.getBoundary(_X,  _Y).isMyArea(x,  y))
return true;
 176 :              }
 177 :              return false;
```

```
178 :        }
                                    .. .. ..
190 :        private void afterMoved() {
191 :               _HitArea.clear();
192 :               for (Block block : _Blocks) {
193 :                      _HitArea.add(block.getBoundary(_X, _Y));
194 :               }
195 :
196 :               if (_OnMoved != null) _OnMoved.onNotify(this);
197 :        }
198 :
                                    .. .. ..
223 : }
```

107-110: 터치가 되었을 때 실행됩니다. 최초에 터치된 곳을 _TouchDown에 저장합니다. _TouchMove는 터치 이후에 이동(Move) 이벤트가 발생할 때마다 해당 위치를 저장합니다.

131-133: 터치 이후에 이동 이벤트가 발생할 때마다 그 위치를 _TouchMove에 저장합니다. _TouchMove의 좌표를 기준으로 Piece가 그려지게 됩니다. 이동 중에 Piece의 좌표는 그대로 두고, _TouchMove를 통해서 현재 이동 중인 곳의 좌표를 사용합니다. 그 이유는 만약 Piece를 이동하다가 잘 못된 공간에 내려 놓으면 원래의 위치로 돌아가거나, 또는 이동 중에 충돌이나 또는 게임의 종료 처리가 되지 않도록 해야 하기 때문입니다.

112-133: Piece를 이동하다가 내려 놓게 되는 동작입니다.

113-114: 내려진 Piece의 픽셀 단위의 좌표 (cx, cy)를 구하고 있습니다. 이때, (Global.blockSize / 2)만큼 더하는 이유는 바둑판 모양의 격자에서 중앙을 표시하기 위해서 입니다.

116-117: 최초의 터치 위치와 이동 중 위치를 초기화 합니다.

119-123: 화면의 범위를 벗어나지 못하도록 합니다.

125-126: 픽셀 단위 좌표에서 바둑판 모양의 좌표로 변경합니다. 정수형태로 나눠지면서 가장 가까운 바둑판 격자의 위치로 이동됩니다.

128: 실제 위치를 변경합니다.

138-172: 실제 터치 이벤트를 처리하는 부분입니다. 상황에 따라서, 이미 설명한 doActionDown, doActionUp, doActionMove를 실행합니다.

173-178: 지정된 좌표가 내(Piece) 영역인지를 확인합니다.

190-198: Piece가 이동되면 HitArea 안의 Boundary 위치를 다시 지정합니다. 그리고, OnMoved 이벤트를 발생하여 외부 리스너에게 알려줍니다.

193: block.getBoundary(_X, _Y)에서 (_X, Y)의 의미는 모든 Boundary가 상대 좌표를 사용하고 있어서, Piece의 현재 좌표를 더해주어야지만 실제 좌표가 되기 때문입니다.

[소스 7] Block.java

```
  1 : package app.main;
  2 :
  3 : import ryulib.graphic.Boundary;
  4 : import android.graphics.Point;
  5 :
  6 : public class Block {
  7 :
  8 : public Block(int x, int y) {
  9 :       super();
 10 :
 11 :       _Point.set(x, y);
 12 :       updateBoundary();
 13 : }
 14 :
 15 : private Point _Point = new Point();
 16 : private Boundary _Boundary = new Boundary(1, 1, Global.blockSize-2,
Global.blockSize-2);
 17 :
 18 : public Point getPoint() {
 19 :       return _Point;
 20 : }
 21 :
 22 : public int getX() {
 23 :       return _Point.x;
 24 : }
 25 :
 26 : public int getY() {
 27 :       return _Point.y;
 28 : }
 29 :
 30 : private void updateBoundary() {
 31 :       _Boundary.setBoundary(
 32 :             (_Point.x * Global.blockSize) + 1,
 33 :             (_Point.y * Global.blockSize) + 1,
 34 :             ((_Point.x+1) * Global.blockSize) - 2,
 35 :             ((_Point.y+1) * Global.blockSize) - 2
 36 :       );
 37 : }
 38 :
 39 : public void decX(int value) {
 40 :       _Point.x = _Point.x - value;
 41 :       updateBoundary();
 42 : }
 43 :
 44 : public void decY(int value) {
 45 :       _Point.y = _Point.y - value;
 46 :       updateBoundary();
 47 : }
 48 :
 49 : public Boundary getBoundary() {
 50 :       return _Boundary;
 51 : }
```

```
52 :
53 : private Boundary _BoundaryCopy = new Boundary(_Boundary);
54 :
55 : public Boundary getBoundary(int x, int y) {
56 :         _BoundaryCopy.setBoundary(_Boundary);
57 :         _BoundaryCopy.incLeft(x * Global.blockSize);
58 :         _BoundaryCopy.incTop (y * Global.blockSize);
59 :
60 :         return _BoundaryCopy;
61 : }
62 :
63 : }
```

조각을 이루는 작은 정사각형의 기능을 담당합니다.

30-37: 위치가 변동될 때마다 실제 좌표를 계산하게 됩니다.

3. 예제 소스 다운받기

예제 소스는 http://ryulib.tistory.com/121 에서 다운 받으실 수 있습니다. 실행 동영상도 같이 있으니 참고하시기 바랍니다. 지금까지는 게임 엔진을 이해하기 위해서, 간단한 테스트용 게임을 제작하는 것을 목표로 하였습니다. 이제, 다음 연재부터는 좀더 게임다운 모양새를 갖추기 위한 내용을 다룰 예정입니다.