게임으로 배우는 안드로이드 개발

글 류종택 ryujt658@hanmail.net (프로그래밍에 관한 질문은 받지 않습니다)
http://ryulib.tistory.com/
(트위터: @RyuJongTaek)

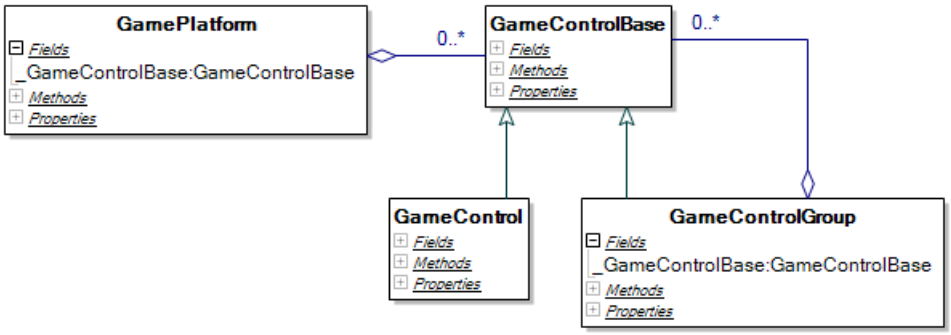
순서

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

심플하고 가벼운 안드로이드 게임 엔진 “게쪽” 소개

1. 게임 엔진 “게쪽”의 개요

“게쪽”은 제가 안드로이드 강좌를 위해서 만든 가벼운 게임 엔진입니다. 원래 이름은 “**게임 엔진 이라고 하기에는 심하게 쪽 팔리지만**” 이며, 이름에서 말해주듯이 심플하고 단순한 라이브러리 모음 입니다.



[그림 1] 게쪽의 Class Diagram

[그림 1]은 “게쪽”의 가장 중요한 클래스의 Class Diagram 입니다. 쉽게 설명하기 위해서 상당히 단순화 하였습니다. 디자인 패턴에 익숙하신 분들은 오른쪽이 컴포지트(Composite) 패턴으로 이루어 져 있다는 것을 아실 수 있을 것 입니다.

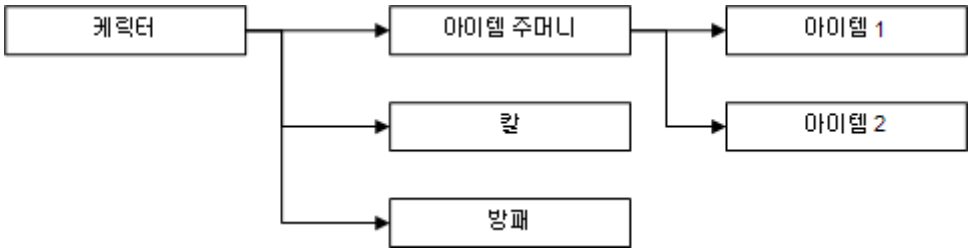
GamePlatform 클래스는 SurfaceView를 상속받아 확장하였습니다. 화면 그리기에 대한 기능을 제공하며, 게임 컨트롤을 관리합니다.

GameControlBase 클래스는 게임 컨트롤의 추상 클래스입니다. 게임 컨트롤이란, 게임에 사용되는 비행기, 미사일 등의 객체를 뜻 합니다. 애니메이션 효과 및 충돌 테스트 등의 기능이 내장되어 있습니다.

그런데, GameControlBase는 GameControlGroup과 GameControl로 나눠지고 있습니다. 컴포지트 패턴이 무엇인지 모르시는 분들께서는 GameControlGroup을 폴더라고 생각하시고, GameControl을 파일이라고 생각하시면 됩니다.

즉, GameControlGroup은 GameControlGroup과 GameControl을 포함 할 수 있다는 뜻 입니다. 예를 들어 게임 케릭터가, [그림 2]와 같이, 하나의 객체가 아닌 여러 개의 객체로 구성되어 있다고 합시다. 케릭터와 아이템 주머니는 다른 컨트롤을 포함해야 하므로 GameControlGroup을 상

속 받아서 확장하고, 칼, 방패, 아이템1, 아이템2는 GameControl을 상속 받아서 확장하면 됩니다.



[그림 2] 케릭터 객체의 구성

이렇게 GameControlGroup과 GameControl로 나눠서 작업하는 이유는, 관리해야 할 객체가 많아질 때, 분류를 통해서 보다 효율적으로 처리하기 위해서 입니다. MP3 파일 수 천 개를 하나의 폴더에 몽땅 저장해 놓은 후, 원하는 가수의 노래만 찾아야 하는 경우를 생각해보시면, 왜 컴포지트 패턴을 사용했는지 이해하기 쉬울 것 입니다. 대부분 그렇게 많은 수의 MP3 파일을 다룰 때는 어떤 분류 기준에 따라 폴더 별로 따로 관리하시고 계실 겁니다. 같은 이치입니다.

2. RyuLib 설치

우선 소스는 아래의 사이트에서 다운받으시면 됩니다. SVN을 사용합니다.

http://dev.naver.com/projects/ryulib4android/

만약 SVN 사용이 익숙하지 않으신 분들께서는 아래의 링크에서 첨부파일을 받아서 사용하시면 됩니다.

http://ryulib.tistory.com/105

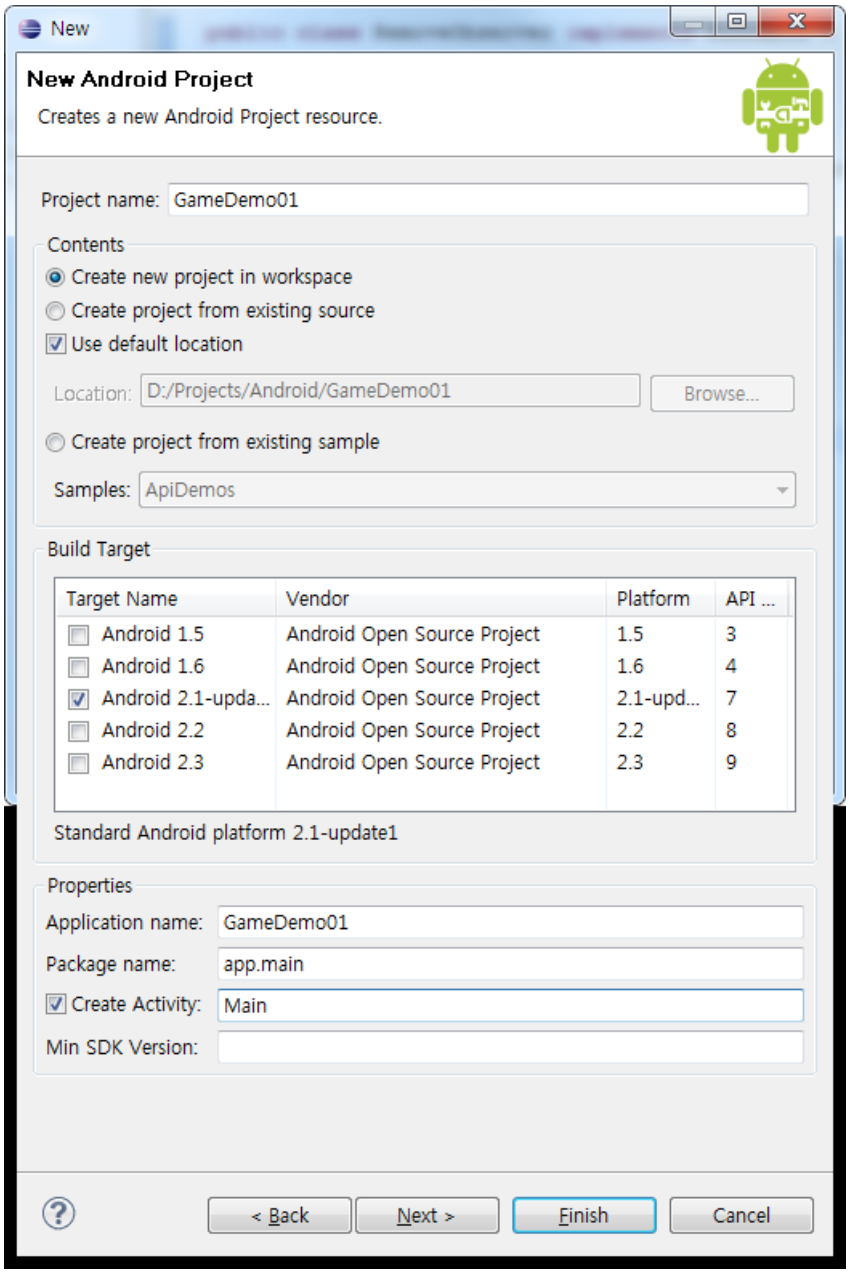
외부 라이브러리로 연결해서 사용하는 것이 어려우신 분들은 당분간 프로그램을 작성하실 때, 소 ryulib 폴더 전체를 프로그램의 src 폴더 밑에 복사해서 사용하시면 됩니다. 저도 연재를 진행하는 동안에는 ryulib 폴더를 src 폴더에 복사해서 사용하도록 하겠습니다.

3. 간단한 도형 출력의 예

우선 "게쪽"이 제공하는 기능들을 이해하는 데 도움이 되도록, 간단한 예제를 들어가며 설명을 하도록 하겠습니다.

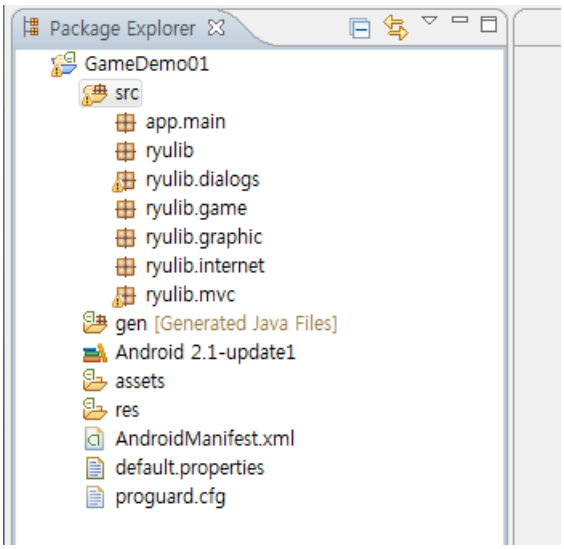
첫 번째 예제는 간단한 도형(사각형)을 출력하는 예제를 작성해보도록 하겠습니다.

우선 메뉴에서 "File → New → Android Project"를 실행하시고, [그림 3]과 같이 프로젝트에 대한 세부 사항을 입력합니다. Project name과 Application name은 GameDemo01로 설정하였습니다. 이 부분은 여러분들이 마음에 드는 이름으로 바꾸어도 문제가 되지 않습니다. Build Target은 2.1을 대상으로 하였으나, 이것 역시 여러분들이 설치하고자 하는 스마트폰의 버전에 맞춰서 설정하셔도 됩니다. Package name 또한 여러분들께서 마음대로 변경하셔도 상관없습니다. Create Activity는 Main으로 설정하였으며, 이 역시 여러분들 마음대로 변경하여 사용하셔도 됩니다. 하지만, 혼선을 피하기 위해서 예제를 따라 하실 때에는 저와 똑 같은 방법으로 입력하시기 바랍니다.



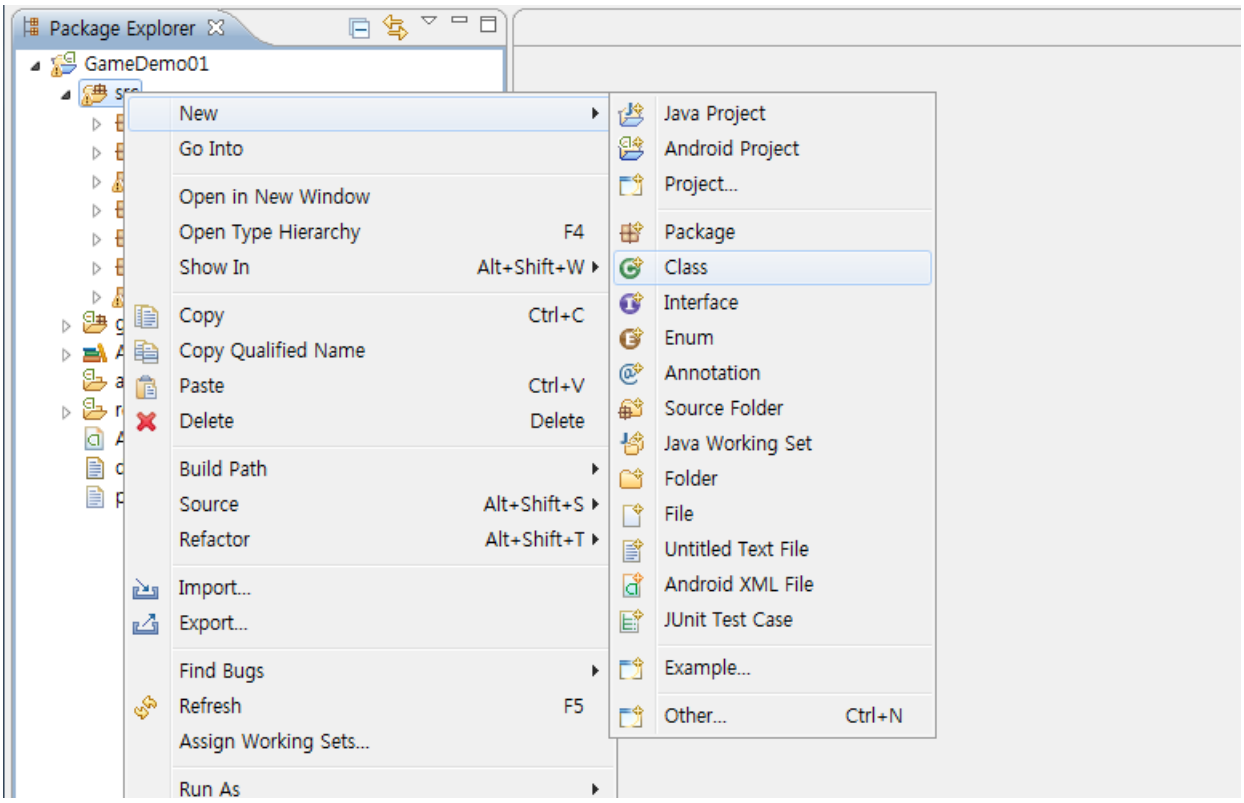
[그림 3] 새로운 프로젝트 생성

이후, [그림 4]과 같이 RyuLib를 복사해 옵니다. MS-Windows를 사용하는 것을 기준으로, 탐색기에서 ryulib 폴더 전체를 복사해서 이클립스의 src 폴더에 붙여넣기를 하시면 됩니다.



[그림 4] RyuLib 복사

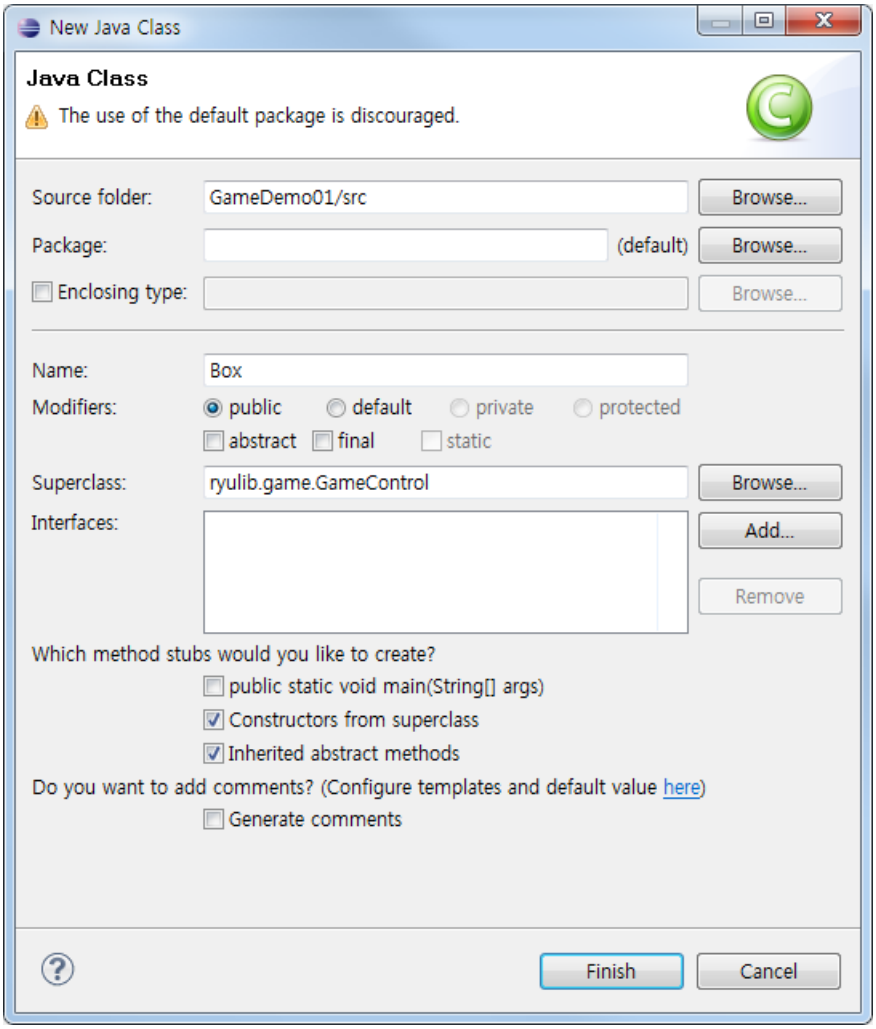
이제 도형(사각형)을 표시 할 클래스를 생성하기 위해서 app.main 패키지 위에서 마우스 오른쪽 버튼을 클릭하시고, [그림 5]와 같이 메뉴를 클릭합니다.



[그림 5] 새로운 클래스 생성

다음으로는 [그림 6]에서처럼, 클래스의 Name을 Box라고 입력하시고, Superclass는 ryulib.game.GameControl을 지정 합니다. 게임에서 화면에 표시되는 모든 객체는 GameControl 클래스를 상속받아야 합니다.

그리고, Constructors from superclass를 선택하시고, Finish 버턴을 클릭합니다.



[그림 6] 클래스 정보 입력

이제 Box.java와 Main.java를 [소스 1]과 [소스 2]와 같이 코딩하시고, 프로그램을 실행하시면 [그림 7]과 같은 화면이 출력됩니다.

[그림 7] 프로그램 실행 결과

이때, CPU 사용률을 보면, 별것 아닌 프로그램이 상당히 많은 CPU를 사용하고 있는 것을 발견하게 됩니다. 화면에는 가만히 있는 것처럼 보이지만, 반복해서 빨간 박스를 그리고 있는 것이기 때문입니다. 이처럼, "게쪽"은 Direct-X 프로그래밍과 같이, 전체 화면을 계속 갱신하도록 되어 있습니다. CPU를 효율적으로 사용하는 방법에 대해서는 추후 다루도록 하겠습니다.

[소스 1] Box.java

```
package app.main;

import ryulib.game.GameControl;
import ryulib.game.GameControlGroup;
import ryulib.game.GamePlatformInfo;
import android.graphics.Canvas;
import android.graphics.Paint;
import android.graphics.Rect;

public class Box extends GameControl {

public Box(GameControlGroup gameControlGroup) {
        super(gameControlGroup);
}

private Rect _rect = new Rect(0, 0, 32, 32);

@Override
protected void onDraw(GamePlatformInfo platformInfo) {
        Paint paint = platformInfo.getPaint();
        Canvas canvas = platformInfo.getCanvas();

```

```
23 :        paint.setARGB(255, 255, 0, 0);
24 :        canvas.drawRect(_rect, paint);
25 : }
26 : }
```

Box는 GameControl을 상속 받아서 확장하여 코딩 합니다. 이때, GameControl의 부모 클래스인 GameControlBase의 메소드를 재정의(override)하면서 게임 엔진이 제공하는 기능을 이용하게 됩니다.

18-25: GameControlBase의 onDraw 메소드를 재정의 하고 있습니다.

20-21: 파라메터로 전달되는 platformInfo 객체에는 GamePlatform이 제공하는 여러 가지 정보가 담겨 있습니다. 그중에서 Paint와 Canvas 객체를 가져옵니다.

Paint와 Canvas는, 굳이 레퍼런스를 옮기지 않아도 됩니다. 즉, 아래와 같이 표현해도 됩니다.
platformInfo.getPaint().setARGB(255, 255, 0, 0);
platformInfo.getCanvas().drawRect(_rect, platformInfo.getPaint());

23: paint의 색상을 빨간색으로 지정합니다.

24: 화면에 지정된 색상으로 박스를 그립니다. 박스의 크기는 16: 라인에서 지정되어 있습니다.

[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 :    /** Called when the activity is first created. */
11 :    @Override
12 :    public void onCreate(Bundle savedInstanceState) {
13 :        super.onCreate(savedInstanceState);
14 :
15 :        _GamePlatform = new GamePlatform(this);
16 :        _GamePlatform.setLayoutParams(
17 :              new LinearLayout.LayoutParams(
18 :                          ViewGroup.LayoutParams.FILL_PARENT,
19 :                          ViewGroup.LayoutParams.FILL_PARENT,
20 :                          0.0F
21 :              )
22 :        );
23 :
24 :        setContentView(_GamePlatform);
25 :
```

```
26 :         _GamePlatform.AddControl(_Box);
27 :      }
28 :
29 :     private GamePlatform _GamePlatform = null;
30 :     private Box _Box = new Box(null);
31 : }
```

15: 게임 엔진의 화면을 담당하는 GamePlatform 생성 합니다.

16-22: GamePlatform을 화면에 꽉 차도록 레이아웃 설정을 합니다.

24: setContentView() 메소드를 이용하여 생성된 GamePlatform이 화면에 표시 되도록 합니다.

26: 게임 컨트롤인 Box를 GamePlatform에 추가합니다. GamePlatform은 추가된 모든 컨트롤을 차례대로 반복하면서 화면에 그려줍니다.

4. 게임 컨트롤 이동 하기

우선 [소스 3]의 23: 라인과 같이 Main.java에 추가 합니다. 이제부터 키보드 이벤트를 받아들이겠다는 의미입니다. 기본으로는 setUseKeyEvent(false)로 되어 있습니다.

[소스 3] Main.java 수정

```
 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 :     /** Called when the activity is first created. */
11 :     @Override
12 :     public void onCreate(Bundle savedInstanceState) {
13 :         super.onCreate(savedInstanceState);
14 :
15 :         _GamePlatform = new GamePlatform(this);
16 :         _GamePlatform.setLayoutParams(
17 :                 new LinearLayout.LayoutParams(
18 :                             ViewGroup.LayoutParams.FILL_PARENT,
19 :                             ViewGroup.LayoutParams.FILL_PARENT,
20 :                             0.0F
21 :                 )
22 :         );
23 :         _GamePlatform.setUseKeyEvent(true);
24 :
```

```
 25 :        setContentView(_GamePlatform);
 26 :
 27 :        _GamePlatform.AddControl(_Box);
 28 :     }
 29 :
 30 :    private GamePlatform _GamePlatform = null;
 31 :    private Box _Box = new Box(null);
 32 : }
```

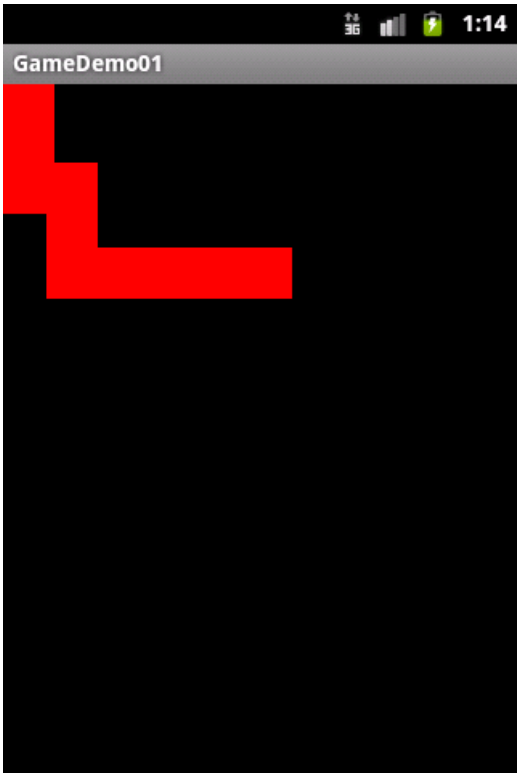
[소스 4] Box.java 수정

```
  1 : package app.main;
  2 :
  3 : import ryulib.game.GameControl;
  4 : import ryulib.game.GameControlGroup;
  5 : import ryulib.game.GamePlatformInfo;
  6 : import android.graphics.Canvas;
  7 : import android.graphics.Paint;
  8 : import android.graphics.Rect;
  9 : import android.view.KeyEvent;
 10 :
 11 : public class Box extends GameControl {
 12 :
 13 : public Box(GameControlGroup gameControlGroup) {
 14 :        super(gameControlGroup);
 15 : }
 16 :
 17 : private int _X = 0;
 18 : private int _Y = 0;
 19 :
 20 : @Override
 21 : protected void onDraw(GamePlatformInfo platformInfo) {
 22 :        Paint paint = platformInfo.getPaint();
 23 :        Canvas canvas = platformInfo.getCanvas();
 24 :
 25 :        paint.setARGB(255, 255, 0, 0);
 26 :
 27 :        Rect _rect = new Rect(_X, _Y, _X+32, _Y+32);
 28 :        canvas.drawRect(_rect, paint);
 29 : }
 30 :
 31 : @Override
 32 : protected  boolean  onKeyDown(GamePlatformInfo  platformInfo,  int
keyCode, KeyEvent msg) {
 33 :        switch (keyCode) {
 34 :               case KeyEvent.KEYCODE_DPAD_LEFT: _X--; break;
 35 :               case KeyEvent.KEYCODE_DPAD_RIGHT: _X++; break;
 36 :               case KeyEvent.KEYCODE_DPAD_UP:
 37 :               case KeyEvent.KEYCODE_Q: _Y--; break;
 38 :               case KeyEvent.KEYCODE_DPAD_DOWN:
 39 :               case KeyEvent.KEYCODE_W: _Y++; break;
 40 :        }
 41 :
 42 :        return false;
 43 :     }
 44 :
 45 : }
```

31-43: 키보드를 이용하여 박스를 움직이기 위해서 추가된 코드입니다. GameControlBase에서 상속받은 onKeyDown 메소드를 재정의하여 사용합니다.

34-39: 각 방향키에 맞춰서 (_X, _Y) 좌표를 변경하고 있습니다.

42: 현재는 false를 리턴 하고 있습니다. 이 부분이 true가 된다면, 더 이상 키보드에 관련된 이벤트 처리는 되지 않고 멈춰버리게 됩니다. 따라서, 더 이상 onKeyDown 키보드 이벤트 처리를 하지 않아도 된다면, true를 리턴 하시면 됩니다.



[그림 8] 박스 이동

[그림 8]은 결과 화면입니다. 몇 가지 문제점이 있습니다. 첫 번째는 너무 느리다는 것입니다. 두 번째는 이동하면서 흔적을 남기는 것입니다. 세 번째는 이동 속도가 일정하지 않다는 것 입니다.

너무 느린 문제는 (_X, _Y) 좌표의 변화 폭을 늘려주면 되지만, 흔적은 매번 그리기 전에 뒤 배경을 지워줘야 합니다. 그리고, 속도가 일정하지 않은 문제는 키보드 이벤트의 발생에 기준에서 컨트롤을 이동하지 말고, 시간에 맞춰서 이동해줘야 합니다.

이러한 것들을 감안해서 다시 소스를 수정해보도록 하겠습니다.

우선 [그림 4]에서 app.main 패키지 위에서 오른쪽 마우스를 클릭하시고, "New → Class" 메뉴를 실행합니다. Name 항목에 "Background"를 입력하시고, "Finish" 버턴을 클릭 합니다. 이후 [소스 5]와 같이 코드를 완성해 주시기 바랍니다.

[소스 5] Background.java

```
  1 : package app.main;
  2 :
  3 : import ryulib.game.GameControl;
  4 : import ryulib.game.GameControlGroup;
  5 : import ryulib.game.GamePlatformInfo;
  6 : import android.graphics.Canvas;
  7 : import android.graphics.Paint;
  8 :
  9 : public class Background extends GameControl {
 10 :
 11 : public Background(GameControlGroup gameControlGroup) {
 12 :        super(gameControlGroup);
 13 :
 14 : }
 15 :
 16 : @Override
 17 : protected void onDraw(GamePlatformInfo platformInfo) {
 18 :        Paint paint = platformInfo.getPaint();
 19 :        Canvas canvas = platformInfo.getCanvas();
 20 :
 21 :        paint.setARGB(255, 0, 0, 0);
 22 :        canvas.drawRect(0,  0,  canvas.getWidth(),  canvas.getHeight(),
paint);
 23 : }
 24 : }
```

21: 라인에서 검은색으로 paint를 지정하고, 22: 라인에서 화면 전체를 채웁니다.

[소스 6] Main.java 수정

```
  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 :     /** Called when the activity is first created. */
 11 :     @Override
 12 :     public void onCreate(Bundle savedInstanceState) {
 13 :         super.onCreate(savedInstanceState);
 14 :
 15 :         _GamePlatform = new GamePlatform(this);
 16 :         _GamePlatform.setLayoutParams(
 17 :               new LinearLayout.LayoutParams(
 18 :                           ViewGroup.LayoutParams.FILL_PARENT,
 19 :                           ViewGroup.LayoutParams.FILL_PARENT,
 20 :                           0.0F
```

```
 21 :                )
 22 :        );
 23 :        _GamePlatform.setUseKeyEvent(true);
 24 :
 25 :        setContentView(_GamePlatform);
 26 :
 27 :        _GamePlatform.AddControl(_Background);
 28 :        _GamePlatform.AddControl(_Box);
 29 :     }
 30 :
 31 :    private GamePlatform _GamePlatform = null;
 32 :    private Background _Background = new Background(null);
 33 :    private Box _Box = new Box(null);
 34 : }
```

27: Background 객체를 추가합니다. 추가된 순서는 나중에 변경하는 방법도 있지만, 다른 모든 게임 컨트롤 보다 먼저 Background 객체가 추가되어야 배경이 먼저 지워지고, 그 위에 다른 게임 컨트롤들이 그 위에 그려지게 됩니다.

이제 프로그램을 실행해 보면, 배경이 계속 지워지면서 움직이는 흔적이 보이지 않게 됩니다. 이제, 움직이는 속도를 일정하면서도 원하는 만큼 조절하는 방법에 대해서 알아보도록 하겠습니다.

[소스 7] Box.java

```
  1 : package app.main;
  2 :
  3 : import ryulib.game.GameControl;
  4 : import ryulib.game.GameControlGroup;
  5 : import ryulib.game.GamePlatformInfo;
  6 : import ryulib.game.JoyStick;
  7 : import android.graphics.Canvas;
  8 : import android.graphics.Paint;
  9 : import android.view.KeyEvent;
 10 :
 11 : public class Box extends GameControl {
 12 :
 13 : public Box(GameControlGroup gameControlGroup) {
 14 :        super(gameControlGroup);
 15 : }
 16 :
 17 : private JoyStick _JoyStick = new JoyStick(250);
 18 :
 19 : @Override
 20 : protected void onDraw(GamePlatformInfo platformInfo) {
 21 :        Paint paint = platformInfo.getPaint();
 22 :        Canvas canvas = platformInfo.getCanvas();
 23 :
 24 :        paint.setARGB(255, 255, 0, 0);
 25 :
 26 :        int x = _JoyStick.getX();
 27 :        int y = _JoyStick.getY();
 28 :
 29 :        canvas.drawRect(x, y, x+32, y+32, paint);
```

```
 30 :
 31 :        _JoyStick.tick(platformInfo.getTick());
 32 : }
 33 :
 34 : @Override
 35 : protected   boolean   onKeyDown(GamePlatformInfo   platformInfo,   int
keyCode, KeyEvent msg) {
 36 :        _JoyStick.onKeyDown(platformInfo, keyCode, msg);
 37 :        return false;
 38 :    }
 39 :
 40 : @Override
 41 : protected   boolean   onKeyUp(GamePlatformInfo   platformInfo,   int
keyCode, KeyEvent msg) {
 42 :        _JoyStick.onKeyUp(platformInfo, keyCode, msg);
 43 :        return false;
 44 :    }
 45 :
 46 : }
```

소스가 전반적으로 변경되어 따로 변경된 부분을 표시하지는 않았습니다. 우선, JoyStickInterface 클래스를 상속받은 JoyStick 클래스가 게임 엔진에 포함되어 있습니다. 이것은 예제로서 제공된 클래스이며, 이것을 참고로 여러분들이 원하는 요구사항을 따로 JoyStickInterface 클래스를 상속받아서 구현하시면 됩니다.

17: JoyStick 객체를 생성하면서 속도는 초당 250 픽셀로 지정하였습니다.

26-27: 박스를 그리기 전에, 현재의 (x, y) 좌표를 얻어오고 있습니다.

31: JoyStick 객체의 상태에 따라서 (x, y) 좌표를 이동합니다. GameControlBase.onDraw() 메소드의 GamePlatformInfo 객체에는 게임 엔진이 제공하는 여러 가지 정보가 있습니다. 그중에서 getTick()은 onDraw() 메소드가 실행되는 간격을 ms(천분의 1초) 단위로 알려줍니다. _JoyStick.tick() 메소드는 그 간격을 이용하여 지정된 속도만큼씩만 (x, y) 좌표가 변화되도록 합니다.

36: 키보드가 눌려졌음을 알려서, JoyStick 객체 내부 상태를 바꿔줍니다. 결국 방향 설정이 됩니다.

37: 키보드가 떼졌음을 알려서, onKeyDown()에서 처리한 내용을 취소해야 함을 JoyStick 객체에게 알려줍니다.

이제 프로그램을 실행해보면, 움직임이 일정하게 (초당 250 픽셀) 움직이는 것을 확인하실 수가 있습니다.

만약 방향 센서를 통해서 안드로이드 장비를 기울여서 박스를 움직이고자 한다면, [소스 8]과 같이 Box.java에 코드를 추가해 주시면 됩니다.

[소스 8] Box.java 코드 추가

```
1 : @Override
2 : protected void onStart(GamePlatformInfo platformInfo) {
3 :        _JoyStick.PrepareOrientationSensor(
4 :                     platformInfo.getGamePlatform().getContext()
5 :        );
6 : }
```

3: 방향 센서를 준비시킵니다.

4: Context 객체를 넘겨줘야 하는데, 이것은 GamePlatformInfo 객체에 포함되어 있는 게임 엔진의 플랫폼에 해당되는 객체에서부터 구해 올 수 있습니다. 또는 여러분들이 Context 객체의 레퍼런스를 따로 관리하셔서 넘겨줘도 됩니다.

이제 프로그램을 실행하시고, 여러분들의 장비(또는 스마트 폰)를 기울여보시면, 박스가 기울임에 따라서 이동하는 것을 확인하실 수가 있습니다.

이때, 장비를 기울일 때마다 화면이 전환되기 때문에 사용이 불편해 집니다. 이럴 때는 다음과 같이, AndroidManifest.xml에서 activity 부분을 수정해주시면 가로 방향으로 고정 됩니다.

```
<activity android:name=".Main" android:label="@string/app_name"
android:screenOrientation="landscape">
```

5. 충돌 테스트

우선 우리가 사용하는 게임 엔진에서는 충돌을 테스트 하기 위해서 ryulib.graphic.Boundary라는 클래스를 사용합니다. Rect와 마찬가지로 네모 상자의 영역을 표시하는 역할을 합니다. 그리고, 이것을 리스트 형태로 묶어서 사용합니다. 현재 만들고 예제의 Box의 경우에는 네모 상자 하나면 충분하지만, 테트리스 등의 게임을 구현 할 때는 상자가 여러 개 있어야 하기 때문에, 엔진 쪽에서는 네모 상자를 리스트로 묶어서 그 중 하나라도 충돌하면 전체가 충돌 한 것으로 간주 합니다. Boundary(네모 상자)의 리스트를 관리하는 클래스는 ryulib.game.HitArea 입니다.

이제, [소스 9]와 같이 Box.java를 수정합니다. 충돌과 관련해서 변화가 없는 부분들을 생략하였습니다.

[소스 9] Box.java 수정

```
   1 : ....
   2 :
   3 : public class Box extends GameControl {
   4 :
   5 : public Box(int x, int y) {
   6 :        super(null);
   7 :
   8 :        _JoyStick.setX(x);
   9 :        _JoyStick.setY(y);
  10 :
  11 :        _HitArea.add(_HitBoundary);
  12 : }
  13 :
  14 : ....
  15 :
  16 : private Boundary _HitBoundary = new Boundary(0, 0, 0, 0);
  17 : private HitArea _HitArea = new HitArea();
  18 :
  19 : @Override
  20 : protected HitArea getHitArea() {
  21 :        _HitBoundary.setBoundary(
  22 :                     _JoyStick.getX(), _JoyStick.getY(),
  23 :                     _JoyStick.getX()+32, _JoyStick.getY()+32
  24 :        );
  25 :
  26 :        return _HitArea;
  27 : }
  28 :
  29 : ....
  30 :
  31 : @Override
  32 : protected void onDraw(GamePlatformInfo platformInfo) {
  33 :        Paint paint = platformInfo.getPaint();
  34 :        Canvas canvas = platformInfo.getCanvas();
  35 :
  36 :        paint.setARGB(255, 255, 0, 0);
  37 :
  38 :        int x = _JoyStick.getX();
  39 :        int y = _JoyStick.getY();
  40 :        _JoyStick.tick(platformInfo.getTick());
  41 :
  42 :        canvas.drawRect(x, y, x+32, y+32, paint);
  43 :
  44 :        GameControl gameControl = this.checkCollision(this);
  45 :        if (gameControl != null) {
  46 :              this.delete();
  47 :              gameControl.delete();
  48 :        }
  49 : }
  50 :
  51 : @Override
  52 : protected   boolean   onKeyDown(GamePlatformInfo   platformInfo,   int
keyCode, KeyEvent msg) {
  53 :        _JoyStick.onKeyDown(platformInfo, keyCode, msg);
  54 :        return true;
  55 :    }
  56 :
  57 : @Override
  58 : protected   boolean   onKeyUp(GamePlatformInfo   platformInfo,   int
keyCode, KeyEvent msg) {
```

```
59 :         _JoyStick.onKeyUp(platformInfo, keyCode, msg);
60 :         return true;
61 :     }
62 :
63 : }
```

5: 생성자의 파라메터를 변경시킨 것에 유의하시기 바랍니다. 박스가 초기에 (0, 0) 좌표에 표시되고 있었는데, 여러 개의 박스를 그대로 생성한다면, 모두 같은 좌표에 표시되어 시작하자마자 충돌하는 현상이 발생하게 될 것이기 때문입니다.

11: HitArea에 Boundary 객체 하나를 추가 합니다.

16: 네모 상자 영역을 표시 할 객체를 만들어 냅니다. 초기에는 영역의 좌표가 (0, 0) - (0, 0)으로 되어 있어서, 면적이 0인 객체가 되었습니다.

19-27: 충돌 테스트를 실행 할 때, 게임 엔진에 의해서 자동으로 호출되는 메소드 입니다. 이를 통해서 충돌 테스트를 하려면 "지금 내가 HitArea를 알려 줄테니까, 이 정보를 토대로 검사해라" 라고, 충돌 테스트 로직에게 알려주게 됩니다. 만약 null을 리턴 한다면 충돌하지 않은 것으로 간주합니다.

21-24: 현재 위치에서 박스 크기만큼의 영역을 Boundary 객체에 지정합니다.

44-48: 충돌 검사를 하는 구간입니다.

44: 출동한 객체 하나를 찾아오는 방법입니다. 복수 개의 객체와의 충돌을 검사하는 방법도 있으나, 설명은 나중으로 미루도록 하겠습니다.

45: 충돌한 객체가 null이 아니면 충돌한 것 입니다.

46-47: 충돌한 두 객체를 모두 지워버립니다. 즉, 화면에서 사라집니다.

54, 60: 두 라인에서 보면, 이전과 달리 true를 리턴 하고 있습니다. 이제 더 이상 키보드 이벤트를 처리할 필요 없다는 뜻 입니다. 만약 우리가 박스 객체를 두 개 생성한다면, 리턴이 false 일 때는 두 객체 모두 움직이게 됩니다. 하지만, true를 리턴 하면. "이 이벤트는 내가 썼으니까 다른 객체는 쓰지마"라고 선언하게 됩니다. 따라서, 해당 객체 이외의 다른 객체는 키보드 이벤트를 사용하지 못하고, 결과적으로 움직이지 못하게 됩니다. 여러 객체가 동시에 움직인다면, 평생 서로 충돌 할 일이 없기 때문입니다.

이제 마지막으로 [소스 10]처럼, Main.java를 수정해서 박스를 두 개 생성하여 서로 충돌하는 지, 그리고 충돌하면 화면에서 사라지는 지 살펴보겠습니다.

[소스 10] Main.java 수정

```
 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 :    /** Called when the activity is first created. */
11 :    @Override
12 :    public void onCreate(Bundle savedInstanceState) {
13 :       super.onCreate(savedInstanceState);
14 :
15 :       _GamePlatform = new GamePlatform(this);
16 :       _GamePlatform.setLayoutParams(
17 :             new LinearLayout.LayoutParams(
18 :                         ViewGroup.LayoutParams.FILL_PARENT,
19 :                         ViewGroup.LayoutParams.FILL_PARENT,
20 :                         0.0F
21 :             )
22 :       );
23 :       _GamePlatform.setUseKeyEvent(true);
24 :
25 :       setContentView(_GamePlatform);
26 :
27 :       _GamePlatform.AddControl(_Background);
28 :       _GamePlatform.AddControl(_Box1);
29 :       _GamePlatform.AddControl(_Box2);
30 :     }
31 :
32 :    private GamePlatform _GamePlatform = null;
33 :    private Background _Background = new Background(null);
34 :    private Box _Box1 = new Box(0, 0);
35 :    private Box _Box2 = new Box(64, 64);
36 : }
```

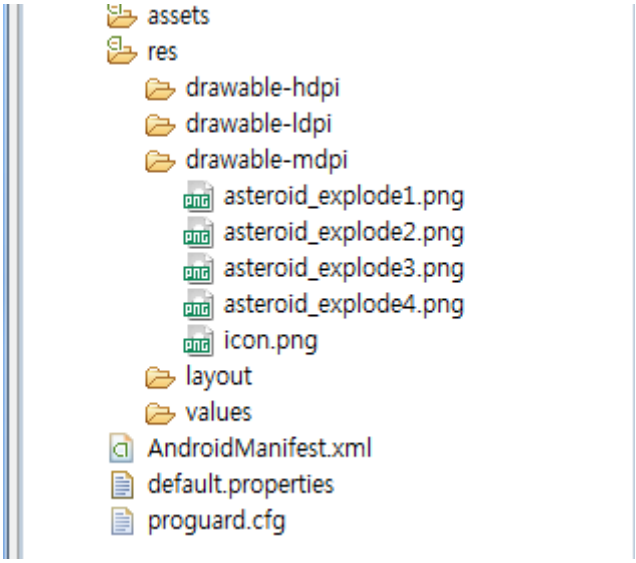
34-35: Box 객체 두 개를 생성하고 있으며, 초기 위치를 각각 다르게 하여 시작하자마자 충돌하는 일이 없도록 하였습니다.

28-29: 생성된 객체들을 게임 플랫폼에 추가시켰습니다.

이제 프로그램을 실행하신 후 키보드를 이용하여 이리 저리 박스를 움직여서 서로 충돌 시켜보시기 바랍니다. 그리고, 두 객체가 화면에서 사라지는 지 확인해보시기 바랍니다.

6. 애니메이션 효과 주기

애니메이션을 위한 이미지를 [그림 9]과 같이 res/drawable-mdpi 폴더에 저장합니다. 일단은 이미지의 개수만 맞으면 상관없습니다. 제가 작성하는 예제와 똑같이 진행하려면, 안드로이드 SDK에 포함된 Sample 중에서 JetBoy 폴더에서 drawable 폴더를 찾아보시면, 같은 이름의 파일들이 있으니 참고하시기 바랍니다. icon.png는 복사하지 않으셔도 자동으로 생성되는 파일입니다.



[그림 9] 이미지 파일 복사

이후, app.main 패키지 위에서 오른쪽 마우스를 클릭하시고, "New → Class" 메뉴를 실행합니다. Name 항목에 "Bang"를 입력하시고, "Finish" 버턴을 클릭 합니다. 이후 [소스 11]과 같이 코드를 완성해 주시기 바랍니다.

[소스 11] Bang.java

```
  1 : package app.main;
  2 :
  3 : import ryulib.game.GameControl;
  4 : import ryulib.game.GameControlGroup;
  5 : import ryulib.game.GamePlatformInfo;
  6 : import android.content.res.Resources;
  7 : import android.graphics.Bitmap;
  8 : import android.graphics.BitmapFactory;
  9 : import android.graphics.Canvas;
 10 : import android.graphics.Paint;
 11 :
 12 : public class Bang extends GameControl {
 13 :
 14 : public Bang(GameControlGroup gameControlGroup) {
 15 :        super(gameControlGroup);
 16 :
```

```
 17 : }
 18 :
 19 : private Bitmap[] _Bitmaps = new Bitmap[4];
 20 : private int _BitmapIndex = 0;
 21 : private long _TickCount = 0;
 22 :
 23 : @Override
 24 : protected void onStart(GamePlatformInfo platformInfo) {
 25 :        Resources resources =
 26 :
      platformInfo.getGamePlatform().getContext().getResources();
 27 :
 28 :        _Bitmaps[0]      =      BitmapFactory.decodeResource(resources,
R.drawable.asteroid_explode1);
 29 :        _Bitmaps[1]      =      BitmapFactory.decodeResource(resources,
R.drawable.asteroid_explode2);
 30 :        _Bitmaps[2]      =      BitmapFactory.decodeResource(resources,
R.drawable.asteroid_explode3);
 31 :        _Bitmaps[3]      =      BitmapFactory.decodeResource(resources,
R.drawable.asteroid_explode4);
 32 : }
 33 :
 34 : @Override
 35 : protected void onDraw(GamePlatformInfo platformInfo) {
 36 :        Paint paint = platformInfo.getPaint();
 37 :        Canvas canvas = platformInfo.getCanvas();
 38 :
 39 :        canvas.drawBitmap(_Bitmaps[_BitmapIndex], 100, 100, paint);
 40 :
 41 :        long tick = platformInfo.getTick();
 42 :
 43 :        _TickCount = _TickCount + tick;
 44 :        if (_TickCount >= 100) {
 45 :               _TickCount = 0;
 46 :               _BitmapIndex++;
 47 :               if (_BitmapIndex >= _Bitmaps.length) {
 48 :                      _BitmapIndex = 0;
 49 :               }
 50 :        }
 51 : }
 52 :
 53 : }
```

19: 애니메이션에 사용할 이미지들을 담을 배열입니다.

25-26: Resources 객체를 가져옵니다.

28-31: Resources 객체로부터 이미지를 가져옵니다. 즉, res/drawable-mdpi 폴더에 저장해두었던 이미지들을 불러옵니다.

39: 현재 _BitmapIndex가 가리키는 이미지를 화면 (100, 100) 좌표에 표시합니다.

41: onDraw()가 호출된 간격(ms 단위)을 가져 옵니다.

44: 간격의 누계가 100ms 가 넘어서면, _BitmapIndex을 증가시켜서 다음 이미지를 화면에 표시합니다.

47-49: _BitmapIndex가 배열의 크기보다 크거나 같으면, 다시 처음으로 되돌립니다.

이제 Main.java를 [소스 12]와 같이 수정해주시고, 프로그램을 실행하시면, 화면에 폭발이 연속으로 이뤄지는 애니메이션을 확인하실 수가 있습니다.

[소스 12] Main.java 수정된 일부 코드

```
  1 :         _GamePlatform.AddControl(_Background);
  2 :         _GamePlatform.AddControl(_Box1);
  3 :         _GamePlatform.AddControl(_Box2);
  4 :         _GamePlatform.AddControl(_Bang);
  5 :      }
  6 :
  7 :     private GamePlatform _GamePlatform = null;
  8 :     private Background _Background = new Background(null);
  9 :     private Box _Box1 = new Box(0, 0);
 10 :     private Box _Box2 = new Box(64, 64);
 11 :     private Bang _Bang = new Bang(null);
```

7. 게임 컨트롤을 그룹화 하기

초반에 이미 설명한 것과 같이, 게임 컨트롤을 유사한 것끼리 묶어서 사용 할 필요가 생길 수 있습니다. 바로 그러한 때에 사용할 수 있는 "게임 컨트롤 그룹화" 기능에 대해서 알아보도록 하겠습니다. 이러한 기능은 게임이 여러 장면으로 나누어져 있을 때 보다 효율적으로 게임 컨트롤을 관리해 줄 수 있도록 합니다.

예를 들어 지금까지 만들어 본 두 가지 "예제 박스를 그리고 움직이고 충돌하기"와 "애니메이션 효과 주기"를 하나의 프로그램에서 장면 전환 형식으로 사용하고자 한다고 가정해 보겠습니다. 처음에는 "박스 움직이기"로 시작했다가, 충돌하면, "애니메이션 효과 주기"가 실행되는 과정을 구현해 볼 수도 있겠지만, 여기서는 간단하게 주석을 통해서 두 그룹의 전환 과정을 보이도록 하겠습니다.

우선, app.main 패키지 위에서 오른쪽 마우스를 클릭하시고, "New → Class" 메뉴를 실행합니다. Name 항목에 "BoxGroup"를 입력하시고, "Finish" 버턴을 클릭 합니다. 이후 [소스 13]과 같이 코드를 완성해 주시기 바랍니다.

[소스 13] BoxGroup.java

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

```
  2 :
  3 : import ryulib.game.GameControlGroup;
  4 :
  5 : public class BoxGroup extends GameControlGroup {
  6 :
  7 : public BoxGroup(GameControlGroup gameControlGroup) {
  8 :        super(gameControlGroup);
  9 :
 10 :        addControl(_Box1);
 11 :        addControl(_Box2);
 12 : }
 13 :
 14 : private Box _Box1 = new Box(0, 0);
 15 :    private Box _Box2 = new Box(64, 64);
 16 :
 17 : }
```

이로써, BoxGroup에는 Box 객체를 두 개 생성하여 보관하게 됩니다.

다시, app.main 패키지 위에서 오른쪽 마우스를 클릭하시고, "New → Class" 메뉴를 실행합니다. Name 항목에 "BangGroup"를 입력하시고, "Finish" 버턴을 클릭 합니다. 이후 [소스 14]과 같이 코드를 완성해 주시기 바랍니다.

[소스 14] BangGroup.java

```
  1 : package app.main;
  2 :
  3 : import ryulib.game.GameControlGroup;
  4 :
  5 : public class BangGroup extends GameControlGroup {
  6 :
  7 : public BangGroup(GameControlGroup gameControlGroup) {
  8 :        super(gameControlGroup);
  9 :
 10 :        addControl(_Bang);
 11 : }
 12 :
 13 :    private Bang _Bang = new Bang(null);
 14 :
 15 : }
```

BangGroup의 경우에는 Bang 객체 하나만을 관리하기 때문에 굳이 GameControlGroup을 이용하여 관리할 필요는 없습니다. 다만, 그룹화에 대한 실습을 위해서 예를 보이는 것 입니다.

이제 [소스 15]와 같이 Main.java를 수정하여 그룹을 통해서 모든 객체가 관리되도록 하겠습니다. Background의 경우에는 모든 그룹이 공통적으로 배경을 지워야 하기 때문에 그룹에 속하지 않고 기존과 같은 방식으로 관리합니다.

[소스 15] Main.java 수정

```
 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 :    /** Called when the activity is first created. */
11 :    @Override
12 :    public void onCreate(Bundle savedInstanceState) {
13 :        super.onCreate(savedInstanceState);
14 :
15 :        _GamePlatform = new GamePlatform(this);
16 :        _GamePlatform.setLayoutParams(
17 :               new LinearLayout.LayoutParams(
18 :                            ViewGroup.LayoutParams.FILL_PARENT,
19 :                            ViewGroup.LayoutParams.FILL_PARENT,
20 :                            0.0F
21 :               )
22 :        );
23 :        _GamePlatform.setUseKeyEvent(true);
24 :
25 :        setContentView( GamePlatform);
26 :
27 : //        _BoxGroup.setVisible(false);
28 :        _BangGroup.setVisible(false);
29 :
30 :        _GamePlatform.AddControl(_Background);
31 :        _GamePlatform.AddControl(_BoxGroup);
32 :        _GamePlatform.AddControl(_BangGroup);
33 :     }
34 :
35 :    private GamePlatform _GamePlatform = null;
36 :    private Background _Background = new Background(null);
37 :    private BoxGroup _BoxGroup = new BoxGroup(null);
38 :    private BangGroup _BangGroup = new BangGroup(null);
39 : }
```

37: BoxGroup 객체를 생성하고 있습니다.

38: BangGroup 객체를 생성하고 있습니다.

27: 주석으로 잠시 가려두었으며, BoxGroup 객체의 Visible 속성을 false로 변경합니다. 만약 주석을 풀게 되면 BoxGroup에 속한 것들은 아무것도 보이지 않게 됩니다. 전체가 한꺼번에 관리되는 것 입니다.

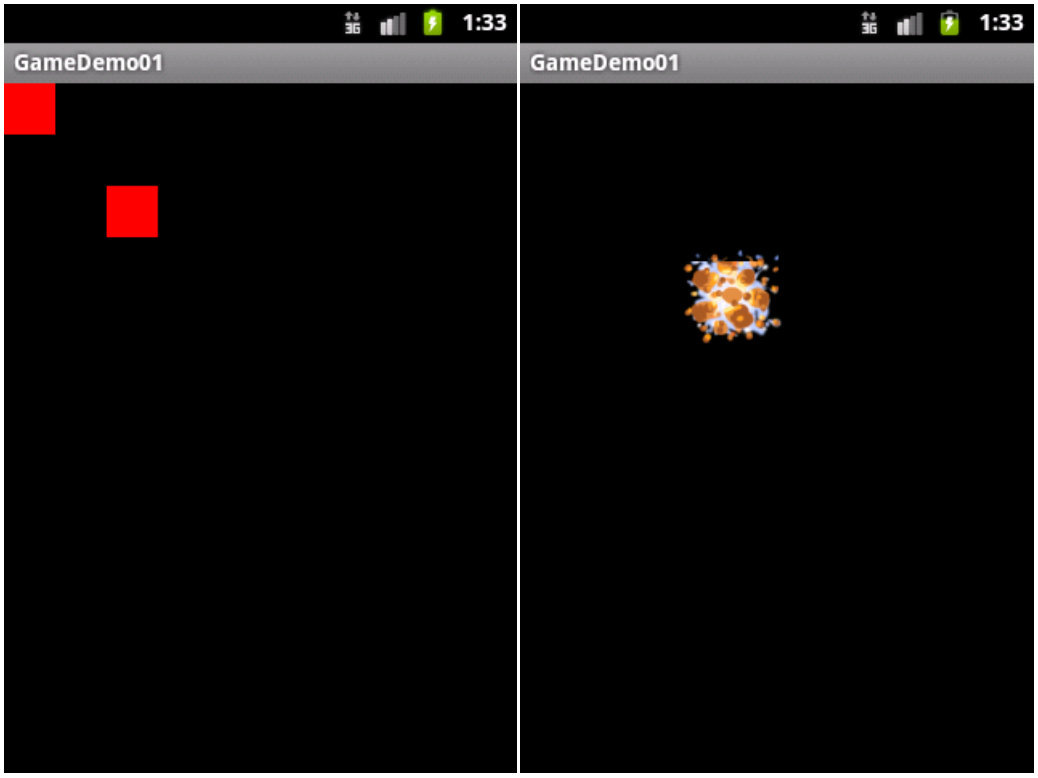
28: BangGroup의 Visible 속성이 false로 변경되어, 그룹에 속한 모든 컨트롤들이 화면에 표시되지 않게 됩니다.

30: 이미 설명 드린 바와 같이, Background 객체는 공통으로 사용하기 때문에 기존과 같은 방식

으로 관리하고 있습니다.

31-32: 생성된 BoxGroup 객체와 BangGroup 객체를 GamePlatform 객체에 등록하여 관리되도록 합니다.

27: 라인과 28: 라인의 주석을 서로 번 갈아서 적용해보면 [그림 10]과 같이 한 쪽 그룹에 속한 컨트롤만 표시되는 것을 확인하실 수가 있습니다.



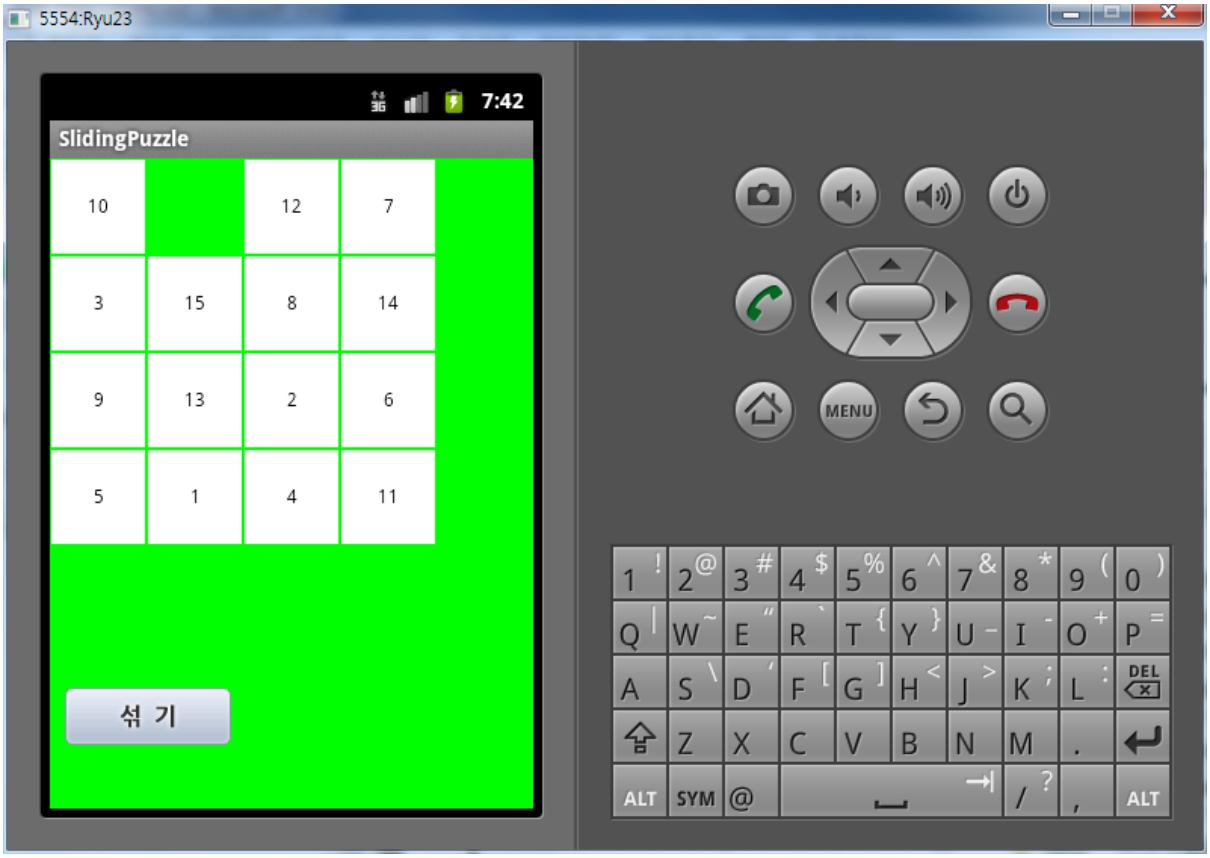
[그림 10] 주석을 번 갈아서 적용해본 결과 화면

슬라이딩 퍼즐 만들기

1. 슬라이딩 퍼즐 설계

슬라이딩 퍼즐은 이미 여러 종류가 앱 스토어에도 공개되어 있기 때문에 독자분들께서 이미 알고 있을 것이라고 생각합니다. 숫자 또는 이미지 퍼즐 조각을 순서대로 맞춰서 원래의 모양으로 되돌리는 것을 목표로 하는 게임입니다.

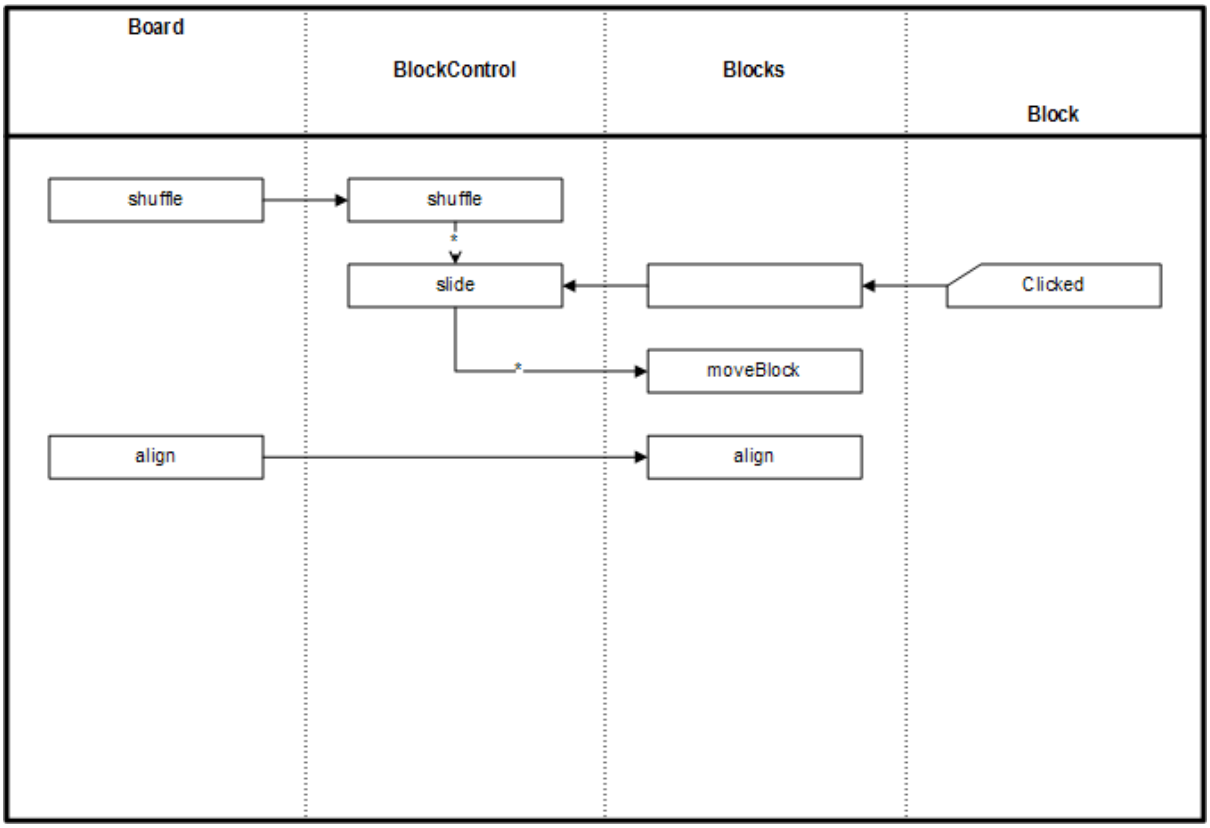
이번 연재에서는 숫자로 되어 있는 퍼즐 조각을 이용하도록 하겠습니다. 이미지를 넣어서 처리하는 과정은 연재가 모두 끝난 후, 제 블로그(http://ryulib.tistory.com/)에 업로드 될 것이니 참고하시기 바랍니다.



[그림 1] 슬라이딩 퍼즐 실행 화면

[그림 2]는 슬라이딩 퍼즐의 전체적인 프로세스의 흐름을 나타낸 것 입니다. 일단 슬라이딩 퍼즐의 가장 중요한 Board 객체는 BlockControl과 Blocks 두 개의 객체로 구성되어 있습니다. 그리고, Blocks는 다시 여러 개의 Block(퍼즐조각) 객체를 관리하게 됩니다.

Board 객체는 shuffle() 메소드를 제공하여, 퍼즐 조각을 무작위로 뒤 섞어 놓을 수가 있습니다. 이때, 실제 퍼즐을 섞는 구현은 BlockControl에 위임하여 분업하도록 하겠습니다. 그리고, 퍼즐 조각에 해당하는 복수의 Block 객체들을 다루는 것보다는 그것을 하나로 묶어서 관리하는 객체인 Blocks를 이용하도록 합니다.

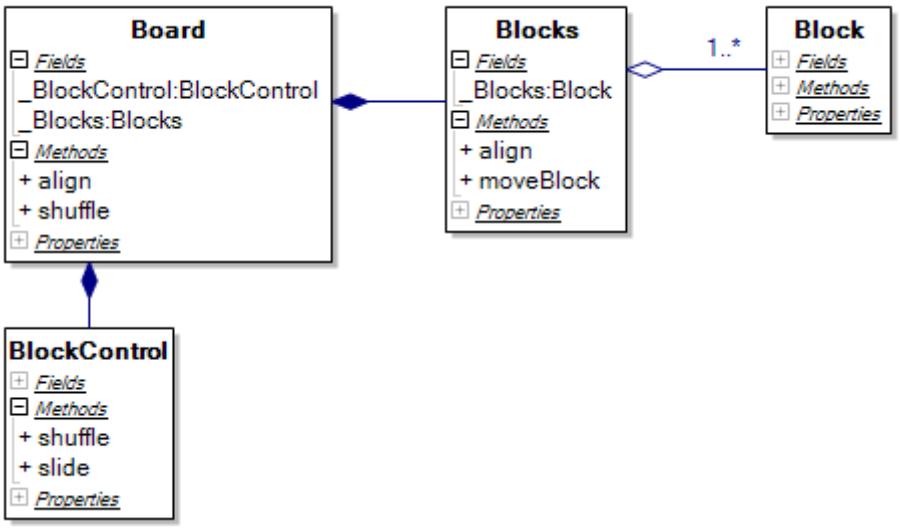


[그림 2] 슬라이딩 퍼즐의 전체적인 프로세스 흐름

[그림 2]의 설명은 아래와 같습니다.

- Board.shuffle() 메소드가 실행되면 BlockControl의 shuffle() 메소드를 실행시켜 실제 퍼즐 조각을 섞어놓는 동작은 BlockControl에서 처리하도록 합니다.
- BlockControl.shuffle()은 무작위로 퍼즐을 선택해서 해당 퍼즐을 공백쪽으로 slide() 메소드를 이용하여 슬라이드 시킵니다.
- BlockControl은 논리적인 구현을 담당하고, 실제 데이터 정보는 Blocks에서 관리하도록 합니다. 따라서, 슬라이드 될 때, 위치 정보가 변경되는 것은 Blocks.moveBlock() 메소드를 이용합니다. 실제 소스에서는 moveBlockLeft(), moveBlockRight() 등으로 이동 메소드가 세분화 되어 있습니다.
- Block 객체에서 클릭 이벤트가 발생하면, BlockControl.slide()를 통해서 클릭된 퍼즐 조각을 스라이드 시킵니다. 다만, Block은 Blocks의 종속된 객체로써 BlockControl과의 의존관계를 갖게 되면, 프로그램이 복잡해집니다. 따라서, Blocks에게 이를 통보하고, 이에 대한 처리는 Blocks가 하게 됩니다. 필자는 이처럼 래밸이 서로 다른 객체끼리의 직접적인 메시지 흐름은 원칙적으로 허용하지 않고 있습니다.
- Board.align() 메소드는 모든 퍼즐 조각들을 제자리에 위치시키도록 정렬합니다.

[그림 3]은 [그림 2]를 통해서 작성한 Class Diagram 입니다. 앞으로 소스를 살펴보시면 알겠지만, 실제 소스는 메소드들이 좀더 보강되어 있습니다. 초기 설계가 상당히 자세하다면 개발에 도움은 되겠지만, 설계에 너무 많은 시간을 사용하는 것은 비효율적이며, 요구사항과 설계는 항상 변경된다는 것이 현실이기 때문에 전반적인 흐름 이상의 것을 파악하기 위해서 시간 낭비하는 것은 좋은 습관이 아닙니다.



[그림 3] 초기 설계에 대한 Class Diagram

2. 실제 소스의 인터페이스 구조

[그림 2]와 [그림 3]과 같이 기본적인 설계를 마치고, 이후 예제를 만들었던 과정을 모두 설명하는 것보다는 최종 결과물을 통해서 설명을 이어가도록 하겠습니다. 따라서, 여기에서 설명되는 인터페이스 구조는 설계과정 이후에 수정된 것이며, 단 번에 설계된 것이 아니고 점진적으로 수정된 것 입니다. 마치 처음에 설계라는 스케치를 한 이후, 조금씩 윤곽이 드러나듯이 작업한 과정의 결과입니다.

[소스 1] IBoard.java

```
  1 : package app.main;
  2 :
  3 : public interface IBoard {
  4 :
  5 : public IBlockControl getBlockControl();
  6 : public IBlocks getBlocks();
  7 :
  8 : public int getBlockSize();
  9 :
 10 : }
```

IBoard는 Board에 대한 인터페이스 부분입니다.

5-6: 라인은 설계를 통해서 발견된 부분이며, 8: 라인의 경우에는 설계 이후 발견된 인터페이스 부분입니다.

5-6: Board의 종속 객체인 BlockControl과 Blocks가 서로 의존적이기 때문에, IBoard를 통해서 다른 객체의 참조를 얻어오기 위해서, getBlockControl()과 getBlocks()를 제공하고 있습니다.

8: 퍼즐 조각의 크기를 알려줍니다. 보다 자세한 설명은 나중으로 미루도록 하겠습니다.

[소스 2] IBlockControl.java

```
  1 : package app.main;
  2 :
  3 : import android.graphics.Point;
  4 :
  5 : public interface IBlockControl {
  6 :
  7 : public void shuffle();
  8 : public void slide(Point point);
  9 :
 10 : }
```

IBlockControl은 BlockControl에 대한 인터페이스 부분입니다.

7: 퍼즐 조각을 섞기 위한 메소드 입니다.

8: point 위치에 있는 퍼즐 조각을 빈 공간 쪽으로 슬라이드 합니다. 퍼즐 조각은 2차원 배열로 보관되어 있기 때문에, 위치를 지정하기 위해서는 (x, y) 좌표가 필요합니다.

[소스 3] IBlocks.java

```
  1 : package app.main;
  2 :
  3 : import ryulib.Direction;
  4 : import android.graphics.Point;
  5 :
  6 : public interface IBlocks {
  7 :
  8 : public void align();
  9 : public void moveBlockLeft(int x, int y);
 10 : public void moveBlockRight(int x, int y);
 11 : public void moveBlockUp(int x, int y);
 12 : public void moveBlockDown(int x, int y);
 13 :
 14 : public int getWidth();
 15 : public int getHeight();
 16 : public Block getBlock(Point point);
 17 : public Direction getBlankDirection(Point point);
 18 : public Point getBlankPosition();
```

```
 19 : }
```

IBlocks는 Blocks에 대한 인터페이스 부분입니다.

8-12: 라인은 설계를 통해서 발견된 부분이며, 14-18: 라인의 경우에는 설계 이후 발견된 인터페이스 부분입니다.

8: 퍼즐 조각을 원 위치로 정렬합니다.

9-12: 퍼즐 조각을 빈 공간 쪽으로 슬라이딩 합니다.

14: 2차원 배열의 가로 크기를 알려줍니다.
15: 2차원 배열의 세로 크기를 알려줍니다.

16: 지정된 위치(point)에 있는 퍼즐 조각을 리턴 합니다.

17: 지정된 위치로부터 빈 공간이 어느 방향에 있는 지 알려줍니다.
18: 빈 공간이 있는 위치를 알려줍니다.

3. 구현 소스 분석

이제 인터페이스가 구현된 실제 소스를 하나씩 설명해보도록 하겠습니다.

[소스 4] Board.java

```
  1 : package app.main;
  2 :
  3 : import android.graphics.Canvas;
  4 : import android.graphics.Paint;
  5 : import ryulib.game.GameControlGroup;
  6 : import ryulib.game.GamePlatformInfo;
  7 :
  8 : public class Board extends GameControlGroup implements IBoard {
  9 :
 10 : public Board(GameControlGroup gameControlGroup) {
 11 :        super(gameControlGroup);
 12 :
 13 :        gameControlGroup.addControl(this);
 14 : }
 15 :
 16 : private BlockControl _BlockControl = new BlockControl(this);
 17 : private Blocks _Blocks = new Blocks(this);
 18 :
 19 : private int _BlockSize = 0;
 20 :
```

```
21 : public void align() {
22 :        _Blocks.align();
23 : }
24 :
25 : public void shuffle() {
26 :        _BlockControl.shuffle();
27 : }
28 :
29 : @Override
30 : protected void onDraw(GamePlatformInfo platformInfo) {
31 :        Paint paint = platformInfo.getPaint();
32 :        Canvas canvas = platformInfo.getCanvas();
33 :
34 :        paint.setARGB(255, 0, 255, 0);
35 :        canvas.drawRect(getBoundary().getRect(), paint);
36 : }
37 :
38 : @Override
39 : public IBlockControl getBlockControl() {
40 :        return _BlockControl;
41 : }
42 :
43 : @Override
44 : public IBlocks getBlocks() {
45 :        return  Blocks;
46 : }
47 :
48 : @Override
49 : public int getBlockSize() {
50 :        return _BlockSize;
51 : }
52 :
53 : public void setBlockSize(int value) {
54 :        _BlockSize =  value;
55 : }
56 :
57 : public void setBoardSize(int width, int height) {
58 :        _Blocks.setSize(width, height);
59 : }
60 :
61 : }
```

16-17: Board는 이미 설명한 것과 같이, 동작 처리를 위한 BlockControl과 블록 조각의 데이터를 관리하는 Blocks으로 구성되어 있습니다.

19: 블록 조각의 픽셀 단위 크기입니다. 이러한 정보는 Blocks에서 관리하여도 되지만, Board 외부에서 블록 조각의 크기를 참조하거나 지정하고자 할 때, 매번 Blocks를 참조하는 번거로움을 피하기 위해서, Board에서 관리하고 있습니다.

21-23: 블록 조각들을 원 위치로 정렬합니다. 실제 구현은 Blocks.align() 메소드에서 처리됩니다.

25-27: 블록 조각을 무작위로 섞어 줍니다. 실제 구현은 BlockControl.shuffle() 메소드에서 처리됩니다.

29-36: 화면에 출력하기 위한 메소드 입니다. 35: 라인에서는, Board의 크기를 나타내는 Boundary의 영역만큼 녹색으로 채우고 있습니다.

38-41: 외부에서 BlockControl의 인터페이스에 대한 참조를 얻고자 할 때 호출 합니다.
43-46: 외부에서 Blocks의 인터페이스에 대한 참조를 얻고자 할 때 호출 합니다.

Board는 이미 거론한 것과 같이 BlockControl과 Blocks로 구성되어 있으며, 이 두 객체들은 서로 의존적입니다. 따라서, 서로의 참조를 알아야 할 필요가 있는데, 서로의 참조는 Owner에 해당하는 Board에서 한꺼번에 관리하기 위해서 getBlockControl()과 getBlocks() 메소드를 제공하는 것입니다. 16-17: 라인에서 보면, BlockControl과 Blocks 객체를 생성 할 때, Board의 레퍼런스를 파라메터로 제공하고 있습니다. 따라서, BlockControl과 Blocks 내부에서는 제공된 Board의 레퍼런스를 이용해서 Board.getBlockControl() 또는 Board.getBlocks()를 호출하여 서로의 레퍼런스를 참조하여 의존적인 메시지를 호출 할 수가 있습니다. 만약 의존 관계에 있는 모든 객체를 생성자의 파라메터로 전달한다면, 설계가 변경되어 의존하는 객체가 변경 될 때마다 소스의 여러 곳을 수정해야 하기 때문에 좋지 못한 방법이라고 할 수 있습니다.

48-51: 블록 조각의 픽셀단위 크기를 리턴 합니다.

53-55: 블록 조각의 크기를 지정합니다. 이것들은 Board의 종속적인 객체들에서 호출하지 않고 있기 때문에 인터페이스를 제공하지 않고 있습니다.

57-59: 퍼즐 조각에 대한 2차원 배열 크기를 지정합니다.

[소스 5] BlockControl.java

```
  1 : package app.main;
  2 :
  3 : import java.util.Random;
  4 :
  5 : import android.graphics.Point;
  6 :
  7 : public class BlockControl implements IBlockControl {
  8 :
  9 : public BlockControl(IBoard board) {
 10 :        super();
 11 :
 12 :        _Board = board;
 13 : }
 14 :
 15 : protected IBoard _Board;
 16 :
 17 : private Random _Random = new Random();
 18 :
 19 : @Override
 20 : public void shuffle() {
 21 :        Point point = new Point();
 22 :
```

```
 23 :        for (int i=0; i<1000; i++) {
 24 :              point.set(
 25 :
       _Random.nextInt(_Board.getBlocks().getWidth()),
 26 :
       _Random.nextInt(_Board.getBlocks().getHeight())
 27 :              );
 28 :
 29 :              if   (_Board.getBlocks().getBlock(point).isBlankBlock()
== false)
 30 :                     slide(point);
 31 :        }
 32 : }
 33 :
 34 : @Override
 35 : public void slide(Point point) {
 36 :        switch (_Board.getBlocks().getBlankDirection(point)) {
 37 :        case Left: moveBlockLeft(point); break;
 38 :        case Right: moveBlockRight(point); break;
 39 :        case Up: moveBlockUp(point); break;
 40 :        case Down: moveBlockDown(point); break;
 41 :        }
 42 : }
 43 :
 44 : private void moveBlockLeft(Point point) {
 45 :        Point blankPosition = _Board.getBlocks().getBlankPosition();
 46 :
 47 :        int blankX = blankPosition.x;
 48 :        int blankY = blankPosition.y;
 49 :        int count = point.x - blankX;
 50 :
 51 :        for (int i=1; i<=count; i++) {
 52 :              _Board.getBlocks().moveBlockLeft(blankX+i, blankY);
 53 :        }
 54 : }
 55 :
 56 : private void moveBlockRight(Point point) {
 57 :        Point blankPosition = _Board.getBlocks().getBlankPosition();
 58 :
 59 :        int blankX = blankPosition.x;
 60 :        int blankY = blankPosition.y;
 61 :        int count = blankX - point.x;
 62 :
 63 :        for (int i=1; i<=count; i++) {
 64 :              _Board.getBlocks().moveBlockRight(blankX-i, blankY);
 65 :        }
 66 : }
 67 :
 68 : private void moveBlockUp(Point point) {
 69 :        Point blankPosition = _Board.getBlocks().getBlankPosition();
 70 :
 71 :        int blankX = blankPosition.x;
 72 :        int blankY = blankPosition.y;
 73 :        int count = point.y - blankY;
 74 :
 75 :        for (int i=1; i<=count; i++) {
 76 :              _Board.getBlocks().moveBlockUp(blankX, blankY+i);
 77 :        }
 78 : }
 79 :
```

```
80 : private void moveBlockDown(Point point) {
81 :        Point blankPosition = _Board.getBlocks().getBlankPosition();
82 :
83 :        int blankX = blankPosition.x;
84 :        int blankY = blankPosition.y;
85 :        int count = blankY - point.y;
86 :
87 :        for (int i=1; i<=count; i++) {
88 :               _Board.getBlocks().moveBlockDown(blankX, blankY-i);
89 :        }
90 : }
91 :
92 : }
```

12: 생성자의 파라메터를 통해서 Board의 객체를 받아오고, 이를 내부의 필드인 _Board에 저장합니다.

19-32: 퍼즐 조각을 섞어 줍니다.

23: 섞는 동작을 1000번 반복합니다.

24-27: 무작위로 퍼즐 조각 하나를 선택하기 위해, Blocks의 크기를 통해서 (x, y) 좌표를 만들어 냅니다. 이때, Blocks의 레퍼런스는 이미 몇 차례 설명한 것과 같이, _Board.getBlocks()을 통해서 얻어 오고 있습니다.

29-30: 선택된 포즐 조각이 빈 공간이 아니라면, 슬라이드를 합니다.

34-42: 선택된 위치의 퍼즐 조각을 슬라이드 합니다. 빈 공간과 선택된 퍼즐의 위치를 파악하여, 어느 방향으로 퍼즐 조각을 슬라이드 할 것인지를 결정하고 있습니다.

44-90: 지정된 퍼즐 조각과 방향으로 슬라이드를 합니다.

각각의 방향으로 슬라이드 하는 메소드들이 서로 유사하기 때문에, moveBlockLeft() 메소드 내부만 설명하도록 하겠습니다.

45: 빈 공간의 위치를 가져 옵니다.
47-48: 빈 공간의 위치를 임시 변수에 저장합니다. `blankPosition.x`, `blankPosition.y`를 직접 사용하지 않는 것은 빈 공간의 위치가 슬라이드 하는 동안 변하기 때문입니다. `blankPosition`는 빈 공간을 처리하는 객체에 대한 레퍼런스라는 것에 유의하셔야 합니다.
49: 선택된 퍼즐 조각과 빈 공간 사이의 거리를 계산합니다. 따라서, 몇 개의 퍼즐 조각을 옮겨야 하는 지를 알게 됩니다. 이 연재의 예제에서는 빈 공간 옆의 퍼즐뿐 아니라, 먼 거리에 있는 퍼즐을 선택해도 선택된 퍼즐 조각과 빈 공간 사이에 있는 모든 퍼즐 조각들이 한꺼번에 이동하게 됩니다.

52: 슬라이드 해야 하는 퍼즐 조각 개수 만큼 퍼즐 조각을 이동합니다.

[소스 6] Blocks.java

```
  1 : package app.main;
  2 :
  3 : import ryulib.Direction;
  4 : import ryulib.OnNotifyEventListener;
  5 : import ryulib.game.GameControlGroup;
  6 : import android.graphics.Point;
  7 :
  8 : public class Blocks implements IBlocks {
  9 :
 10 : public Blocks(IBoard board) {
 11 :        super();
 12 :
 13 :        _Board = board;
 14 : }
 15 :
 16 : private IBoard _Board;
 17 :
 18 : private Block _Blank = null;
 19 : private Block[][] _Blocks = null;
 20 : private int _Width = 0;
 21 : private int _Height = 0;
 22 :
 23 : public void setSize(int width, int height) {
 24 :        _Width = width;
 25 :        _Height = height;
 26 :
 27 :        align();
 28 : }
 29 :
 30 : @Override
 31 : public void align() {
 32 :        _Blocks = new Block[_Width][_Height];
 33 :
 34 :        ((GameControlGroup) _Board).clearControls();
 35 :
 36 :        for (int y=0; y<_Height; y++) {
 37 :               for (int x=0; x<_Width; x++) {
 38 :                      // TODO
 39 :                      _Blocks[x][y]  =  new  Block(((GameControlGroup)
_Board));
 40 :                      _Blocks[x][y].setNo(x + _Width*y + 1);
 41 :                      _Blocks[x][y].setSize(_Board.getBlockSize());
 42 :                      _Blocks[x][y].setPosition(x, y);
 43 :                      _Blocks[x][y].setIsBlankBlock(false);
 44 :                      _Blocks[x][y].setOnClick(_OnBlockClick);
 45 :               }
 46 :        }
 47 :
 48 :        _Blank = _Blocks[_Width-1][_Height-1];
 49 :        _Blank.setIsBlankBlock(true);
 50 : }
 51 :
 52 : @Override
 53 : public int getWidth() {
 54 :        return _Width;
```

```
  55 : }
  56 :
  57 : @Override
  58 : public int getHeight() {
  59 :        return _Height;
  60 : }
  61 :
  62 : @Override
  63 : public Block getBlock(Point point) {
  64 :        return _Blocks[point.x][point.y];
  65 : }
  66 :
  67 : @Override
  68 : public Direction getBlankDirection(Point point) {
  69 :        Direction result = Direction.NoWhere;
  70 :
  71 :        if (point.x == _Blank.getPosition().x) {
  72 :               if   (point.y   >   _Blank.getPosition().y)   result   =
Direction.Up;
  73 :               else result = Direction.Down;
  74 :        } else if (point.y == _Blank.getPosition().y) {
  75 :               if   (point.x   >   _Blank.getPosition().x)   result   =
Direction.Left;
  76 :               else result = Direction.Right;
  77 :        }
  78 :
  79 :        return result;
  80 : }
  81 :
  82 : @Override
  83 : public Point getBlankPosition() {
  84 :        return _Blank.getPosition();
  85 : }
  86 :
  87 : private void swapBlocks(Block a, Block b) {
  88 :        _Blocks[a.getPosition().x][a.getPosition().y] = b;
  89 :        _Blocks[b.getPosition().x][b.getPosition().y] = a;
  90 :
  91 :        int tempX = a.getPosition().x;
  92 :        int tempY = a.getPosition().y;
  93 :        a.setPosition(b.getPosition());
  94 :        b.setPosition(tempX, tempY);
  95 : }
  96 :
  97 : @Override
  98 : public void moveBlockLeft(int x, int y) {
  99 :        if (x > 0) swapBlocks(_Blocks[x][y], _Blocks[x-1][y]);
 100 :        }
 101 :
 102 :        @Override
 103 :        public void moveBlockRight(int x, int y) {
 104 :               if    (x    <    (_Width-1))    swapBlocks(_Blocks[x][y],
_Blocks[x+1][y]);
 105 :        }
 106 :
 107 :        @Override
 108 :        public void moveBlockUp(int x, int y) {
 109 :               if (y > 0) swapBlocks(_Blocks[x][y], _Blocks[x][y-1]);
 110 :        }
 111 :
```

```
112 :        @Override
113 :        public void moveBlockDown(int x, int y) {
114 :               if    (y    <    (_Height-1))    swapBlocks(_Blocks[x][y],
_Blocks[x][y+1]);
115 :        }
116 :
117 :        private    OnNotifyEventListener    _OnBlockClick    =    new
OnNotifyEventListener() {
118 :               @Override
119 :               public void onNotify(Object sender) {
120 :                      Block block = (Block) sender;
121 :
      _Board.getBlockControl().slide(block.getPosition());
122 :               }
123 :        };
124 :
125 : }
```

12: BlockControl과 마찬가지로 생성자의 파라메터를 통해서 Board의 객체를 받아오고, 이를 내부의 필드인 _Board에 저장합니다.

27: Blocks의 크기가 지정되면, align()을 통해서 기존의 퍼즐 조각을 지워버리고, 새로 생성하면서 정렬 시키게 됩니다. align() 메소드는 정렬이라기 보다 새로 만들기에 해당합니다.

30-50: Blocks 크기에 맞춰서 Block(블록 조각) 객체를 생성하고 초기 데이터를 지정합니다.

34: 기존의 블록 조각들을 모두 삭제 합니다.

39: 블록 조각을 생성하고, 생성자의 파라메터를 Board로 지정합니다. 이때, Block은 GameControl을 상속 받은 클래스이기 때문에 GameControlGroup 객체를 파라메터로 넘겨줘야 합니다. Board는 GameControlGroup 클래스를 상속받았기 때문에 Board 레퍼런스를 넘겨주고 있습니다. 이때, 인터페이스인 IBoard가 아닌 GameControlGroup으로 타입을 캐스팅하고 있습니다.
40: 퍼즐 조각의 순서를 지정합니다.
41: 퍼즐 조각의 픽셀 단위 크기를 지정합니다.
42: 퍼즐 조각의 위치를 지정합니다.
43: 빈 공간에 해당하는 지 여부를 지정합니다.
44: 퍼즐 조각이 클릭 되었을 때 처리 할 이벤트 리스너를 지정합니다.

48-49: 마지막에 위치한 퍼즐 조각을 빈 공간으로 지정합니다.

62-65: 지정된 위치(point)에 있는 퍼즐 조각 객체의 레퍼런스를 리턴 합니다.

67-80: 지정된 위치(point)로부터 어느 방향에 빈 공간이 있는 지 알려줍니다.

82-85: 빈 공간의 위치를 알려줍니다.

97-115: 지정된 위치의 퍼즐 조각을 지정된 방향으로 슬라이드 시킵니다. 이동 할 수 없는 위치의 퍼즐 조각이 지정되면 이동을 무시합니다. 실제 이동은 지정된 퍼즐 조각과 지정된 방향에 있는 퍼즐 조각을 서로 바꿔주는 것으로 구현되어 있습니다. 항상 빈 공간 옆에 있는 퍼즐 조각부터 차례로 이동시키고 있기 때문에, 지정된 퍼즐 조각이 빈 공간으로 이동하는 것처럼 보이게 됩니다. 빈 공간도 실제로는 퍼즐 조각과 같은 객체로 처리되어 있다는 것에 유의하시기 바랍니다.

87-95: 슬라이딩 할 때, 퍼즐 조각 두 개의 위치를 변경하는 메소드 입니다. 위치에 해당하는 정보가 _Blocks 배열과 퍼즐 조각 자체 Block 객체의 Position 정보에 중복되어 있기 때문에, 두 가지 정보 모두를 변경하고 있습니다. 하나로 통일해서 작업할 수도 있지만, 그럴 경우에는 연산이 더 많이 필요하기 때문에 중복 처리하였습니다.

117-123: 퍼즐 조각이 클릭되었을 때 처리 할 이벤트 리스너를 구현한 곳 입니다.

121: 현재 클릭된 퍼즐 조각을 슬라이드 하기 위해서, 슬라이드 전문가인 BlockControl 객체에게 구현을 위임하고 있습니다. 모든 소스를 한 곳에서 처리 할 수도 있겠지만, 유사한 코드를 모아두는 편이 훨씬 이해하기 쉽기 때문에 클래스 여러 개로 분업화 하였습니다.

[소스 7] Main.java

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

```
 29 :        _Board = new Board(_GamePlatform.getGameControlGroup());
 30 :        _Board.getBoundary().setBoundary(0, 0, 320, 480);
 31 :        _Board.setBlockSize(64);
 32 :        _Board.setBoardSize(4, 4);
 33 :
 34  :                    _ButtonShuffle  =  new  ryulib.game.ImageButton
(_GamePlatform.getGameControlGroup());
 35 :        _ButtonShuffle.setPosition(10, 350);
 36 :        _ButtonShuffle.setImageUp(R.drawable.up);
 37 :        _ButtonShuffle.setImageDown(R.drawable.down);
 38 :        _ButtonShuffle.setOnClick(_OnShuffle);
 39 :    }
 40 :
 41 : private GamePlatform _GamePlatform = null;
 42 :    private Board _Board = null;
 43 :    private ryulib.game.ImageButton _ButtonShuffle = null;
 44 :
 45  :              private  OnNotifyEventListener  _OnShuffle  =  new
OnNotifyEventListener() {
 46 :        @Override
 47 :        public void onNotify(Object sender) {
 48 :               _Board.shuffle();
 49 :        }
 50 : };
 51 :
 52 : }
```

19: 터치를 통해서 퍼즐 조각을 클릭해야 하기 때문에, 모션 이벤트를 사용합니다.

29-32: 슬라이딩 퍼즐의 기능을 제공하는 Board 객체를 생성하고 초기 데이터를 지정합니다.
30: Board의 크기를 지정합니다.
31: 퍼즐 조각의 크기를 지정합니다.
32: 퍼즐 조각의 배열 크기를 지정합니다. 소스에서는 4x4 조각의 퍼즐로 지정하였습니다.

34-38: 퍼즐 조각을 무작위로 섞어 놓기 위해서 버턴을 사용합니다. 안드로이드에서 기본으로 제공하는 ImageButton이 아닌 것에 유의하시기 바랍니다.
35: 버턴의 위치를 지정합니다.
36: 버턴이 눌러지지 않았을 경우의 이미지를 지정합니다.
37: 버턴이 눌러졌을 경우의 이미지를 지정합니다.
38: 버턴이 클릭 됐을 경우 이벤트 처리를 수행 할 이벤트 리스너를 지정합니다.

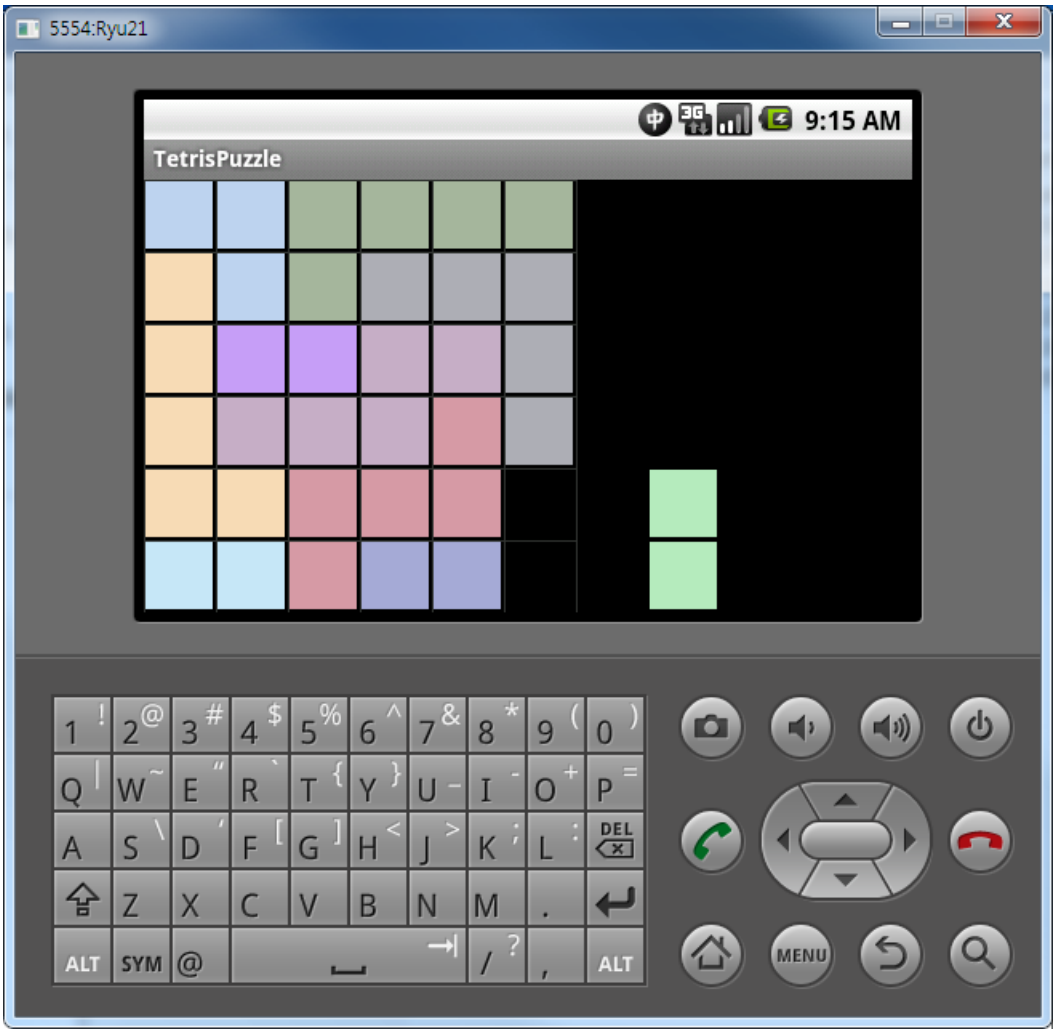
41-50: 버턴이 클릭 됐을 경우 이벤트 처리하는 이벤트 리스너 입니다.
48: 버턴이 클릭되면, Board의 shuffle() 메소드를 실행합니다.

이미지는 프로젝트 소스의 drawable 폴더에 넣어주시면 됩니다. 이미지 파일의 이름은 소문자로 되어 있어야 합니다. [그림 4]는 예제에서 사용 중인 버턴의 이미지입니다.

[그림 4] 예제에서 사용한 평상 시와 눌러졌을 때의 버턴 이미지

테트리스 퍼즐 만들기

1. 테트리스 퍼즐 설계



[그림 1] 테스리스 퍼즐 실행 화면

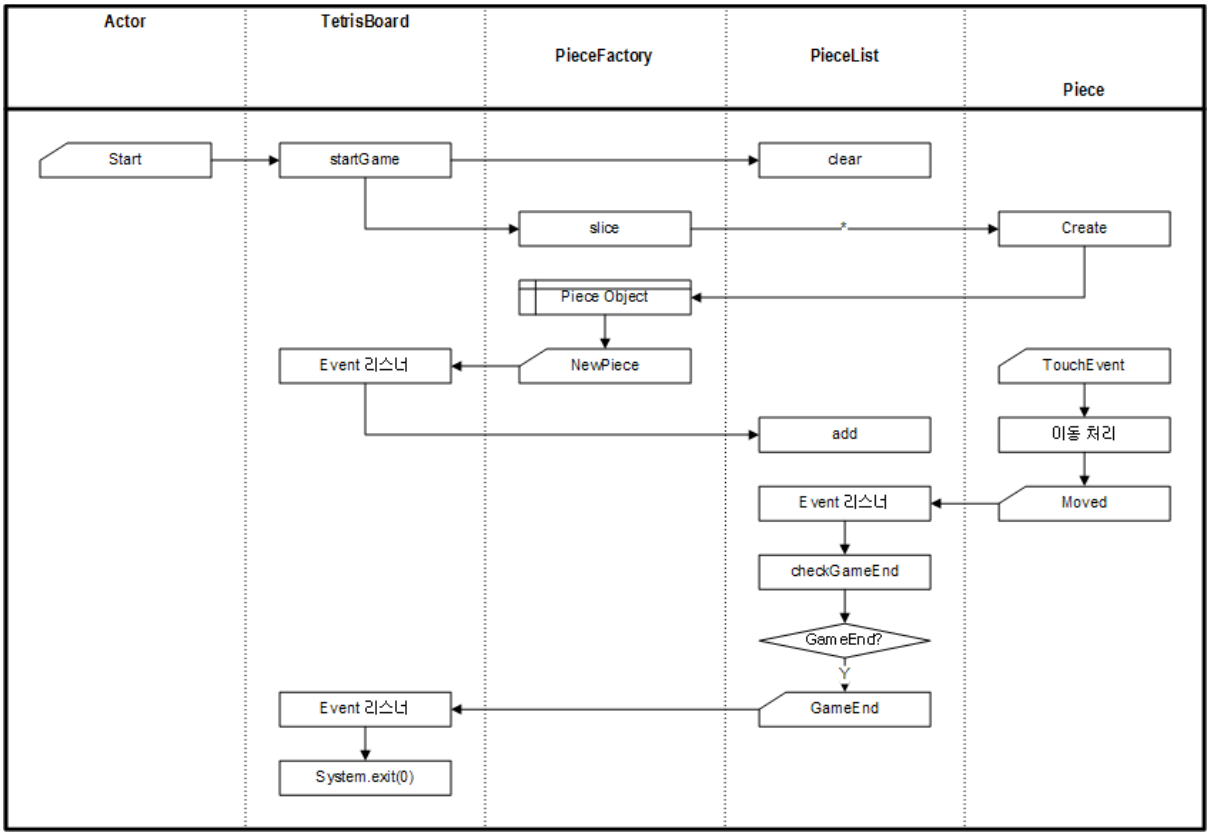
이번에 만들 테트리스 퍼즐은 [그림 1]과 같이 조각난 퍼즐을 다시 하나로 합치는 게임입니다. 이동 중이거나 퍼즐이 겹쳐져 있을 때는 이것을 알 수 있도록 퍼즐 색상을 어둡게(투명) 처리하였습니다.

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.

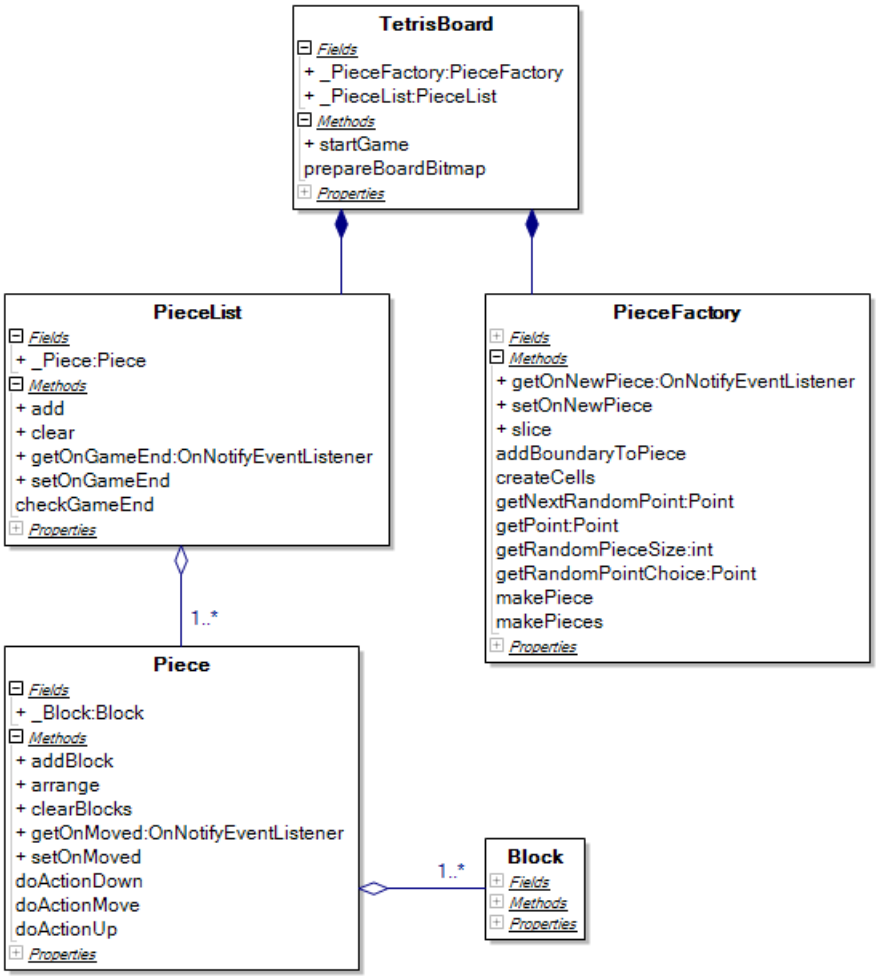


[그림 2] 게임 소스 전반적인 흐름에 대한 Job Flow

- 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 납니

다.

- 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 위

치한 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: 라인에서는 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 : }
```

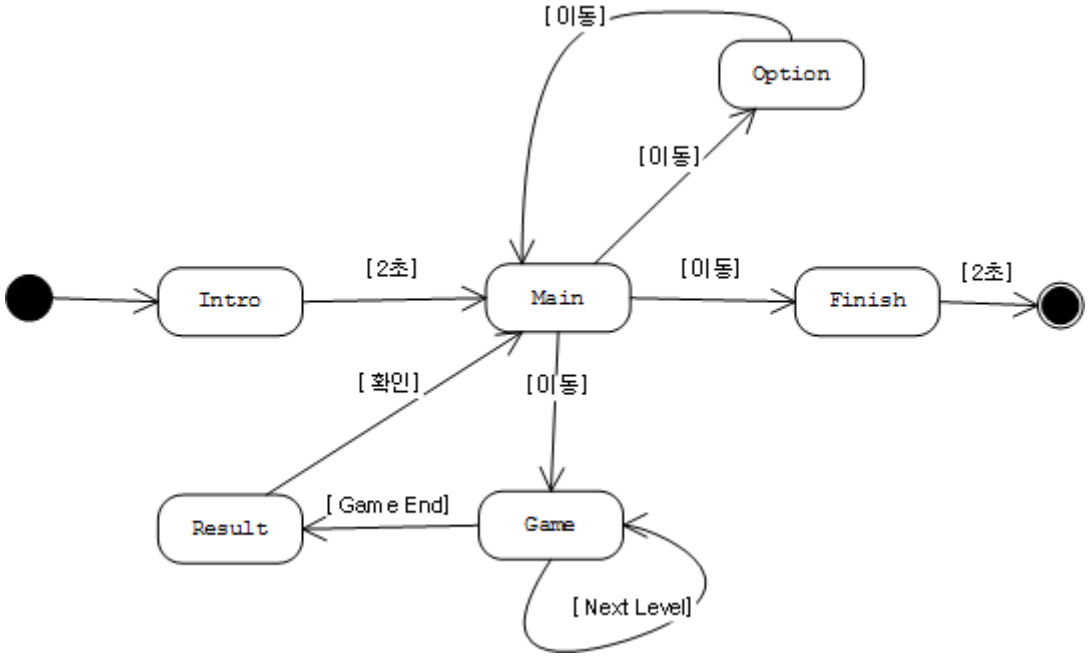
```
43 :
44 : public void decY(int value) {
45 :        _Point.y = _Point.y - value;
46 :        updateBoundary();
47 : }
48 :
49 : public Boundary getBoundary() {
50 :        return _Boundary;
51 : }
52 :
53 : private Boundary _BoundaryCopy = new Boundary(_Boundary);
54 :
55 : public Boundary getBoundary(int 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.

슈팅 게임 만들기 #1

1. 화면 전환

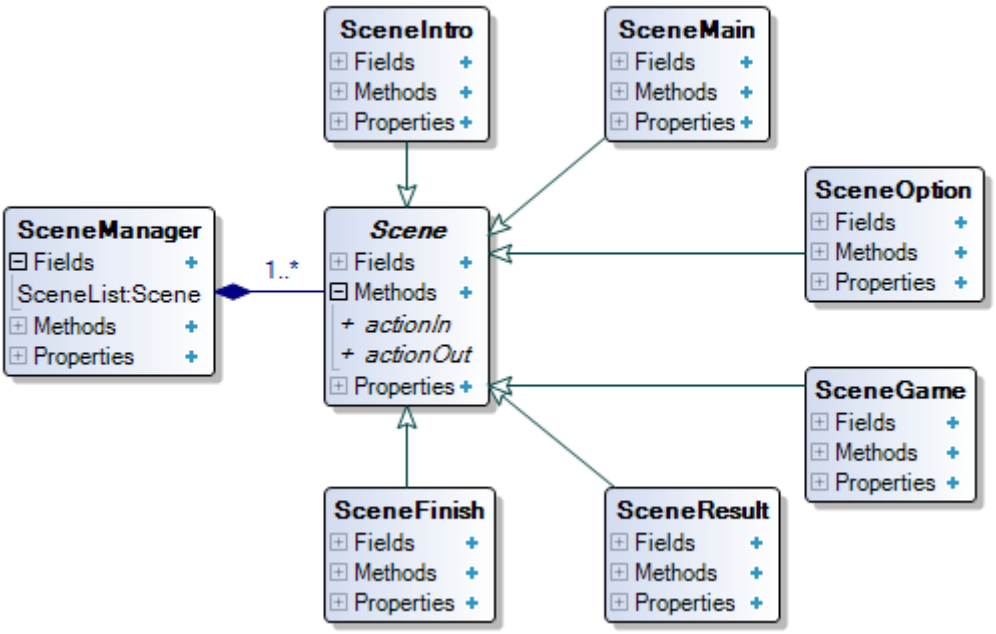


[그림 1] 화면 전환에 대한 상태도

[그림 1]은 게임 전반적이 화면 전환에 대한 상태도입니다. 이에 대한 설명은 아래와 같습니다.

- Intro: 게임이 시작될 때 보여주는 화면.
  - 화면이 보여지고 난 뒤 2초 후면 Main으로 이동 합니다.
  - 실제 게임에서는 필요한 데이터 등을 로딩하는 동안 보여주는 형식을 취합니다.
- Main: 게임의 Main Menu에 해당하는 화면
  - 사용자의 선택에 따라 Option/Game/Finish 화면으로 이동 할 수 있습니다.
- Option: 게임의 옵션을 설정하고자 할 때 이용하는 화면
  - 옵션을 설정한 이후에는 Main 화면으로 이동합니다.
- Game: 게임 화면
  - 게임이 진행되는 화면입니다.
  - 게임은 여러 개의 레벨로 구성되어 있을 수 있으며, 한 레벨을 완료하면 다음 레벨로 자동으로 이동하게 됩니다.
  - 게임 로직에 의해서 게임이 끝나게 되면 게임 결과에 해당하는 Result 화면으로 이동합니다.
- Result: 게임 결과 내용을 표시하는 화면
  - 사용자가 내용을 확인하면, 다시 Main 화면으로 이동합니다.
- Finish: 게임이 종료될 때 보여주는 화면

■ 게임 어플리케이션이 완전히 종료되기 전에 표시되는 화면이며, 2초 정도 표시한 후에 어플리케이션이 종료됩니다.



[그림 2] Class Diagram

[그림 2]는 화면을 표시하기 위하여 작성한 클래스들의 Class Diagram 입니다. 우선 화면마다 클래스를 따로 만들어서 관리할 예정입니다. 그리고, 화면이 복수 개로 존재하다 보니, 이것을 한 곳에서 일관성 있도록 관리하기 위해서 SceneManager를 두어서 접근하려고 합니다.

모든 화면은 자신이 표시될 때 초기화를 처리하는 actionIn()과 자신에게서 다른 화면으로 전환될 때 종료화를 처리하는 actionOut() 메소드를 공통적으로 가지고 있기 때문에, Scene 클래스를 부모 클래스로 하고 이를 상속받아서 처리하도록 하였습니다.

[소스 1] Scene.java

```
  1 : package app.scene;
  2 :
  3 : import ryulib.ValueList;
  4 : import ryulib.game.GameControl;
  5 : import ryulib.game.GameControlGroup;
  6 :
  7 : public class Scene extends GameControl {
  8 :
  9 : public Scene(GameControlGroup gameControlGroup) {
 10 :        super(gameControlGroup);
 11 :
 12 :        setVisible(false);
 13 : }
 14 :
 15 : final void doActionIn(Scene oldScene, ValueList params) {
 16 :        setVisible(true);
```

```
17 :        actionIn(oldScene, params);
18 : }
19 :
20 : public void actionIn(Scene oldScene, ValueList params) {
21 :        // Override 해서 사용
22 : }
23 :
24 : final void doActionOut(Scene newScene) {
25 :        setVisible(false);
26 :        actionOut(newScene);
27 : }
28 :
29 : public void actionOut(Scene newScene) {
30 :        // Override 해서 사용
31 : }
32 :
33 : }
```

15: 이미 설명한 것과 같이 자신이 표시될 때 초기화를 실행하는 메소드 입니다. 파라메터로 이전 화면과 params를 사용합니다. params는 화면 전환 시에 부가 정보가 필요할 경우를 대비해서 작성하였습니다. 현재 진행하는 예제에서는 사용하지 않기 때문에 null로 처리하고 있습니다.

17: 자식 클래스에서 Override 하여 사용하게될 actionIn()을 호출합니다. 각 화면마다 처리할 내용이 달라질 것이 자명하기 때문입니다. 이와 같은 형태를 템플릿 메소드 패턴이라고 합니다.

actionOut()도 actionIn()과 처리가 같기 때문에 설명을 생략합니다. 다만, params가 논리상으로 필요하지 않기 때문에 새로 표시될 화면에 해당하는 newScene 객체만을 파라메터를 전달 받습니다.

[소스 2] SceneManager.java

```
 1 : package app.scene;
 2 :
 3 : import ryulib.ValueList;
 4 : import ryulib.game.GameControlGroup;
 5 :
 6 :
 7 : public class SceneManager {
 8 :
 9 : private static SceneManager _MyObject = new SceneManager();
10 :
11 : private SceneManager() {}
12 :
13 : public static SceneManager getInstance() {
14 :    return _MyObject;
15 : }
16 :
17 : private GameControlGroup _GameControlGroup = null;
18 :
19 : private Scene _Scene = null;
20 : private SceneIntro _SceneIntro = null;
21 : private SceneMain _SceneMain = null;
22 : private SceneOption _SceneOption = null;
```

```
  23 : private SceneGame _SceneGame = null;
  24 : private SceneResult _SceneResult = null;
  25 : private SceneFinish _SceneFinish = null;
  26 :
  27 : public void intro(ValueList params) {
  28 :        if    (_SceneIntro    ==    null)    _SceneIntro    =    new
SceneIntro(_GameControlGroup);
  29 :        setScene(_SceneIntro, params);
  30 : }
  31 :
  32 : public void main(ValueList params) {
  33 :        if    (_SceneMain    ==    null)    _SceneMain    =    new
SceneMain(_GameControlGroup);
  34 :        setScene(_SceneMain, params);
  35 : }
  36 :
  37 : public void option(ValueList params) {
  38 :        if    (_SceneOption    ==    null)    _SceneOption    =    new
SceneOption(_GameControlGroup);
  39 :        setScene(_SceneOption, params);
  40 : }
  41 :
  42 : public void game(ValueList params) {
  43 :        if    (_SceneGame    ==    null)    _SceneGame    =    new
SceneGame( GameControlGroup);
  44 :        setScene(_SceneGame, params);
  45 : }
  46 :
  47 : public void result(ValueList params) {
  48 :        if    (_SceneResult    ==    null)    _SceneResult    =    new
SceneResult(_GameControlGroup);
  49 :        setScene(_SceneResult, params);
  50 : }
  51 :
  52 : public void finish(ValueList params) {
  53 :        if    (_SceneFinish    ==    null)    _SceneFinish    =    new
SceneFinish(_GameControlGroup);
  54 :        setScene(_SceneFinish, params);
  55 : }
  56 :
  57 : public void setGameControlGroup(GameControlGroup value) {
  58 :        _GameControlGroup = value;
  59 : }
  60 :
  61 : public GameControlGroup getGameControlGroup() {
  62 :        return _GameControlGroup;
  63 : }
  64 :
  65 : public void setScene(Scene value, ValueList params) {
  66 :        Scene temp = _Scene;
  67 :        _Scene = value;
  68 :
  69 :        if (temp != null) temp.doActionOut(_Scene);
  70 :        if (_Scene != null) _Scene.doActionIn(temp, params);
  71 : }
  72 :
  73 : public Scene getScene() {
  74 :        return _Scene;
  75 : }
  76 :
```

```
 77 : }
```

각 화면으로 이동하기 위해서는 intro(), main(), option() 등의 메소드를 호출하기만 하면 됩니다. 이 메소드들은 상당히 유사한 코드 형식을 보이고 있기 때문에 중복을 제거하는 것이 좋을 수도 있습니다. 하지만, 화면의 개수가 거의 고정이며, 변화가 된다고 해도 심할 가능성이 없기 때문에 간단하게 표현하였습니다. 변화가 없을 가능성이 높은데 굳이 코드를 복잡하게 작성하는 것이 득이 되지 않을 것이라는 판단입니다.

9-15: SceneManager는 싱글톤 패턴으로 작성되어 있기 때문에 참조할 변수를 사용하지 않고, SceneManager.getInstance()를 호출해서 참조하면 됩니다.

65: 실제 화면을 전환하는 메소드 입니다.

69: 이전 화면이 null 이 아니라면, 이전 화면 객체의 doActionOut() 메소드를 호출 합니다.

70: 현재 표시될 화면 객체의 doActionIn() 메소드를 호출 합니다.

2. Main Activity

[소스 3] 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 : import app.scene.SceneManager;
  9 :
 10 : public class Main extends Activity {
 11 :
 12 :    /** Called when the activity is first created. */
 13 :    @Override
 14 :    public void onCreate(Bundle savedInstanceState) {
 15 :        super.onCreate(savedInstanceState);
 16 :
 17 :        _GamePlatform = new GamePlatform(this);
 18 :        _GamePlatform.setUseMotionEvent(true);
 19 :        _GamePlatform.setLayoutParams(
 20 :              new LinearLayout.LayoutParams(
 21 :                          ViewGroup.LayoutParams.FILL_PARENT,
 22 :                          ViewGroup.LayoutParams.FILL_PARENT,
 23 :                          0.0F
 24 :              )
 25 :        );
 26 :        setContentView(_GamePlatform);
```

```
 27 :
 28 :        SceneManager.getInstance().setGameControlGroup(
 29 :               _GamePlatform.getGameControlGroup());
 30 :        SceneManager.getInstance().intro(null);
 31 :     }
 32 :
 33 : private GamePlatform _GamePlatform = null;
 34 :
 35 : }
```

소스가 지금까지의 Main.java와 유사하기 때문에 틀리 부분만 설명하도록 하겠습니다.

28: 관리하고 있는 화면 객체들이 GameControl을 상속 받고 있기 때문에 GameControlGroup 객체를 지정해줘야 합니다. _GamePlatform.getGameControlGroup()을 지정하여, 모든 화면 객체들이 소속 될 GameControlGroup을 지정합니다.

29: 어플리케이션이 실행되자 마자 intro 화면을 표시합니다. 앞서 설명한 것과 같이 params는 현재의 예제에서는 사용하지 않기 때문에 null을 전달하고 있습니다. 이는 다른 게임에서도 같은 소스를 재사용하기 위해서 미리 만들어둔 인터페이스 입니다.

3. 초기 화면 표시

[그림 3] 초기 화면

[소스 4] SceneIntro.java

```
  1 : package app.scene;
  2 :
  3 : import ryulib.ValueList;
  4 : import ryulib.game.GameControlGroup;
  5 : import ryulib.game.GamePlatformInfo;
  6 : import android.graphics.Bitmap;
  7 : import android.graphics.BitmapFactory;
  8 : import android.graphics.Canvas;
  9 : import android.graphics.Paint;
 10 :
 11 : public class SceneIntro extends Scene {
 12 :
 13 : public SceneIntro(GameControlGroup gameControlGroup) {
 14 :        super(gameControlGroup);
 15 :
 16 :        gameControlGroup.addControl(this);
 17 : }
 18 :
 19 : private Canvas _Canvas = null;
 20 : private Paint  Paint = null;
 21 : private Bitmap _Bitmap = null;
 22 : private long _DisplayTime = 0;
 23 :
 24 : @Override
 25 : public void actionIn(Scene oldScene, ValueList params) {
 26 :        _DisplayTime = 2000;
 27 : }
 28 :
 29 : @Override
 30 : public void actionOut(Scene oldScene) {
 31 :        _Bitmap = null;
 32 : }
 33 :
 34 : @Override
 35 : protected void onStart(GamePlatformInfo platformInfo) {
 36 :        _Canvas = platformInfo.getCanvas();
 37 :        _Paint = platformInfo.getPaint();
 38 : }
 39 :
 40 : @Override
 41 : protected void onDraw(GamePlatformInfo platformInfo) {
 42 :        _DisplayTime = _DisplayTime - platformInfo.getTick();
 43 :        if (_DisplayTime <= 0) {
 44 :              SceneManager.getInstance().main(null);
 45 :              return;
 46 :        }
 47 :
 48 :        if (_Bitmap == null) {
 49 :              _Bitmap = BitmapFactory.decodeResource(
 50 :
      platformInfo.getGamePlatform().getContext().getResources(),
 51 :                              app.main.R.drawable.begin);
 52 :        }
 53 :
 54 :        _Canvas.drawBitmap(_Bitmap, 0, 0, _Paint);
 55 : }
 56 :
 57 : }
```

26: 표시될 시간을 2초로 설정합니다.

31: 현재 화면이 다른 화면으로 전환될 때 _Bitmap 객체 참조를 삭제합니다.

42-46: _DisplayTime을 화면 표시 간격인 getTick() 만큼씩 줄여나가다가, 2초가 지난 시점에서 main 화면으로 전환합니다.

48: _Bitmap 객체가 null이면 초기화면의 이미지를 불러들입니다.

54: 초기화면 이미지를 [그림 3]과 같이 표시합니다.

4. 메인 화면 표시

[그림 4] 메인 화면

메인 화면에서는 option, game, finish 화면으로 이동할 수 있으며, 이를 위해서 버턴 3개를 통해서 이동하도록 하였습니다. 버턴은 ryulib.game.ImageButton 클래스를 사용합니다.

[소스 5] SceneMain.java

```
  1 : package app.scene;
  2 :
  3 : import ryulib.OnNotifyEventListener;
  4 : import ryulib.ValueList;
  5 : import ryulib.game.GameControlGroup;
```

```
  6 : import ryulib.game.GamePlatformInfo;
  7 : import ryulib.game.ImageButton;
  8 : import android.graphics.Bitmap;
  9 : import android.graphics.BitmapFactory;
 10 : import android.graphics.Canvas;
 11 : import android.graphics.Paint;
 12 : import android.graphics.Typeface;
 13 : import app.main.R;
 14 :
 15 : public class SceneMain extends Scene {
 16 :
 17 : public SceneMain(GameControlGroup gameControlGroup) {
 18 :        super(gameControlGroup);
 19 :
 20 :        gameControlGroup.addControl(this);
 21 : }
 22 :
 23 : private Canvas _Canvas = null;
 24 : private Paint _Paint = null;
 25 : private Bitmap _Bitmap = null;
 26 :
 27 : private ImageButton _btOption = null;
 28 : private ImageButton _btGame = null;
 29 : private ImageButton _btExit = null;
 30 :
 31 : @Override
 32 : public void actionIn(Scene oldScene, ValueList params) {
 33 :        _btOption = new ImageButton(getGameControlGroup());
 34 :        _btOption.getPaint().setTextSize(16);
 35 :        _btOption.getPaint().setTypeface(Typeface.DEFAULT_BOLD);
 36 :        _btOption.setCaption("Option");
 37 :        _btOption.setPosition(235, 24);
 38 :        _btOption.setImageUp(R.drawable.btn_up);
 39 :        _btOption.setImageDown(R.drawable.btn_dn);
 40 :        _btOption.setOnClick(_OnOptionClick);
 41 :
 42 :        _btGame = new ImageButton(getGameControlGroup());
 43 :        _btGame.getPaint().setTextSize(16);
 44 :        _btGame.getPaint().setTypeface(Typeface.DEFAULT_BOLD);
 45 :        _btGame.setCaption("Start Game");
 46 :        _btGame.setPosition(235, 24 + 80*1);
 47 :        _btGame.setImageUp(R.drawable.btn_up);
 48 :        _btGame.setImageDown(R.drawable.btn_dn);
 49 :        _btGame.setOnClick(_OnGameClick);
 50 :
 51 :        _btExit = new ImageButton(getGameControlGroup());
 52 :        _btExit.getPaint().setTextSize(16);
 53 :        _btExit.getPaint().setTypeface(Typeface.DEFAULT_BOLD);
 54 :        _btExit.setCaption("Exit");
 55 :        _btExit.setPosition(235, 24 + 80*2);
 56 :        _btExit.setImageUp(R.drawable.btn_up);
 57 :        _btExit.setImageDown(R.drawable.btn_dn);
 58 :        _btExit.setOnClick(_OnExitClick);
 59 : }
 60 :
 61 : @Override
 62 : public void actionOut(Scene newScene) {
 63 :        _Bitmap = null;
 64 :
 65 :        _btOption.delete();
```

```
 66 :        _btOption = null;
 67 :
 68 :        _btGame.delete();
 69 :        _btGame = null;
 70 :
 71 :        _btExit.delete();
 72 :        _btExit = null;
 73 : }
 74 :
 75 : @Override
 76 : protected void onStart(GamePlatformInfo platformInfo) {
 77 :        _Canvas = platformInfo.getCanvas();
 78 :        _Paint = platformInfo.getPaint();
 79 : }
 80 :
 81 : @Override
 82 : protected void onDraw(GamePlatformInfo platformInfo) {
 83 :        if (_Bitmap == null) {
 84 :               _Bitmap = BitmapFactory.decodeResource(
 85 :
      platformInfo.getGamePlatform().getContext().getResources(),
 86 :                             app.main.R.drawable.main);
 87 :        }
 88 :
 89 :         Canvas.drawBitmap( Bitmap, 0, 0,  Paint);
 90 : }
 91 :
 92 : private      OnNotifyEventListener      _OnOptionClick      =      new
OnNotifyEventListener() {
 93 :        @Override
 94 :        public void onNotify(Object sender) {
 95 :               SceneManager.getInstance().option(null);
 96 :        }
 97 : };
 98 :
 99 : private      OnNotifyEventListener      _OnGameClick      =      new
OnNotifyEventListener() {
100 :               @Override
101 :               public void onNotify(Object sender) {
102 :                      SceneManager.getInstance().game(null);
103 :               }
104 :        };
105 :
106 :        private    OnNotifyEventListener    _OnExitClick    =    new
OnNotifyEventListener() {
107 :               @Override
108 :               public void onNotify(Object sender) {
109 :                      SceneManager.getInstance().finish(null);
110 :               }
111 :        };
112 :
113 : }
```

31-59: 화면의 초기화에서 필요한 버턴 3개를 생성하고 있습니다.

38,47,56: 버턴은 평상시에 해당하는 이미지와
39,48,57: 눌러졌을 때 사용하는 이미지 두 개를 지정하여 사용합니다.

36,45,54: ImageButton 클래스의 setCaption("표시할 문자열") 메소드는 버턴 위에 표시될 문자열을 지정할 수 있도록 합니다.

61-73: 화면의 종료 처리에서는 배경화면과 버턴의 참조를 null로 지정합니다. 사용하지 않는 리소스를 반환하기 위해서 입니다.

92-97: _btOption 버턴이 클릭되면 option 화면으로 이동합니다.

99-104: _btGame 버턴이 클릭되면 game 화면으로 이동합니다.

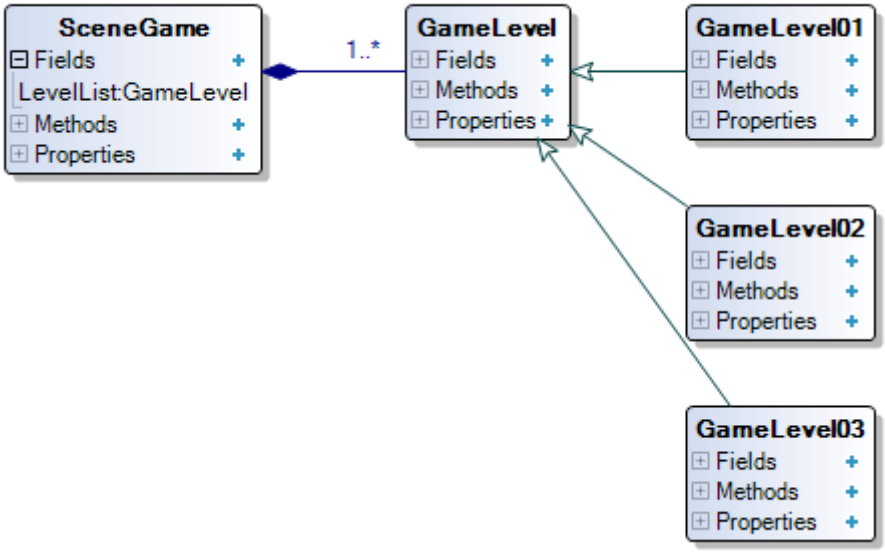
106-111: _Exit 버턴이 클릭되면 finish 화면으로 이동합니다.

[그림 5]는 _ btOption 버턴이 클릭됐을 때의 화면입니다.

[그림 5] 버턴이 눌러졌을 때 (터치)

5. 게임 화면 표시

게임 화면의 경우에는 자체적으로도 레벨에 따른 화면 관리가 필요합니다. 따라서 [그림 6]과 같이 클래스를 구성하도록 하였습니다.



[그림 6] 게임 화면에 대한 Class Diagram

[소스 6] SceneGame.java

```
 1 : package app.scene;
 2 :
 3 : import ryulib.ValueList;
 4 : import ryulib.game.GameControlGroup;
 5 : import app.game.level.GameLevel;
 6 : import app.game.level.GameLevel01;
 7 : import app.game.level.GameLevel02;
 8 : import app.game.level.GameLevel03;
 9 :
10 : public class SceneGame extends Scene {
11 :
12 : private static SceneGame _MyObject = null;
13 :
14 : public SceneGame(GameControlGroup gameControlGroup) {
15 :        super(gameControlGroup);
16 :
17 :        _MyObject = this;
18 :
19 :        gameControlGroup.addControl(this);
20 : }
21 :
22 : public static SceneGame getInstance() {
23 :     return _MyObject;
24 : }
25 :
26 : private GameLevel _GameLevel = null;
27 :
28 : public void setGameLevel(int level) {
29 :        if (_GameLevel != null) {
30 :               _GameLevel.delete();
31 :               _GameLevel = null;
32 :        }
33 :
34 :        switch (level) {
35 :               case         1:          _GameLevel         =         new
```

```
GameLevel01(getGameControlGroup()); break;
  36 :                case         2:         _GameLevel         =         new
GameLevel02(getGameControlGroup()); break;
  37 :                case         3:         _GameLevel         =         new
GameLevel03(getGameControlGroup()); break;
  38 :                default: ; // TODO : raise Exception
  39 :        }
  40 :
  41 :        getGameControlGroup().addControl(_GameLevel);
  42 : }
  43 :
  44 : @Override
  45 : public void actionIn(Scene oldScene, ValueList params) {
  46 :        setGameLevel(1);
  47 : }
  48 :
  49 : @Override
  50 : public void actionOut(Scene newScene) {
  51 :        if (_GameLevel != null) {
  52 :                _GameLevel.delete();
  53 :                _GameLevel = null;
  54 :        }
  55 : }
  56 :
  57 : }
```

22-24: SceneGame의 경우에는 싱글톤 패턴을 응용하여 static 메소드를 통해서 참조하는 것과 변수를 통해서 접근하는 두 가지를 모두 허용하는 형태입니다. 모든 화면의 객체가 SceneManager 내부에서 변수를 통해서 참조되고 있기 때문에 SceneGame도 변수를 통해서 접근하도록 하였습니다. 하지만, 복수의 GameLevel에서 접근할 때, SceneManager를 경유하는 것보다 직접 SceneGame의 객체를 참조하는 것이 직관적이라고 판단하여 static 메소드를 통해서 참조할 수 있도록 하였습니다.

28-42: 지정된 번호(level)에 따라서 GameLevel을 전환해주는 메소드 입니다.

29-32: 현재 레벨의 객체를 삭제합니다.

34-39: 지정된 번호에 맞춰 그에 상응하는 GameLevel 객체를 생성합니다.

41: 생성된 객체를 GameControlGroup에 포함시킵니다. 이로써 GameLevel 객체가 화면에 표시될 준비가 됩니다.

44-47: 게임 화면이 표시될 때 초기화 처리입니다. 첫 번째 GameLevel을 표시합니다.

49-55: 게임 화면의 종료 처리입니다. 생성된 GameLevel 객체를 삭제합니다.

아래의 [소스 7]은 첫 번째 GameLevel에 해당하는 소스입니다. 38: 라인에서 보듯이, 화면이 터치되면 바로 두 번째 GameLevlel로 이동하도록 되어 있습니다. GameLevel02.java의 경우도 똑같

은 원리로 GameLevel03으로 이동하도록 되어 있습니다. GameLevel03에서 화면이 터치되면, result 화면으로 이동하여 게임 결과를 화면에 표시합니다. 이 부분은 다음 연재에서 구체적으로 다룰 예정이며, 실제로는 어느 레벨에서든지 게임이 종료되면 바로 result 화면으로 이동하도록 할 예정입니다. 현재는 전체 화면의 전환을 다루고 있기 때문에 게임의 구체적인 내용은 흉내만 내어 테스트가 가능하도록 하였습니다.

[소스 7] GameLevel01.java

```
   1 : package app.game.level;
   2 :
   3 : import android.graphics.Canvas;
   4 : import android.graphics.Paint;
   5 : import android.view.MotionEvent;
   6 : import app.scene.SceneGame;
   7 : import ryulib.game.GameControlGroup;
   8 : import ryulib.game.GamePlatformInfo;
   9 :
  10 : public class GameLevel01 extends GameLevel {
  11 :
  12 : public GameLevel01(GameControlGroup gameControlGroup) {
  13 :        super(gameControlGroup);
  14 :        // TODO Auto-generated constructor stub
  15 : }
  16 :
  17 : private Canvas _Canvas = null;
  18 : private Paint _Paint = null;
  19 :
  20 : @Override
  21 : protected void onStart(GamePlatformInfo platformInfo) {
  22 :        _Canvas = platformInfo.getCanvas();
  23 :        _Paint = platformInfo.getPaint();
  24 : }
  25 :
  26 : @Override
  27 : protected void onDraw(GamePlatformInfo platformInfo) {
  28 :        _Paint.setARGB(255, 0, 0, 0);
  29 :        _Canvas.drawRect(0,           0,           _Canvas.getWidth(),
_Canvas.getHeight(), _Paint);
  30 :
  31 :        _Paint.setARGB(255, 255, 255, 255);
  32 :        _Canvas.drawText("GameLevel01...", 100, 100, _Paint);
  33 : }
  34 :
  35 : @Override
  36 :       protected boolean onTouchEvent(GamePlatformInfo platformInfo,
MotionEvent event) {
  37 :        if (event.getAction() == MotionEvent.ACTION_DOWN) {
  38 :               SceneGame.getInstance().setGameLevel(2);
  39 :        }
  40 :
  41 :        return true;
  42 :     }
  43 :
  44 : }
```

6. 마무리

[그림 6] 종료 화면, 2초간 표시되고 어플리케이션이 종료됩니다.

설명되지 않은 option과 finish 화면은 각각 main과 intro 화면과 유사한 소스이기 때문에 생략하였습니다.

슈팅 게임 만들기 #2

1. 게임 관련 패키지 추가

이번 연재의 소스에는 아래와 같이, 두 개의 패키지가 추가되었습니다. 이 패키지에 있는 소스들은 http://ryulib.tistory.com/ 에서 "Jet Boy 따라하기"라는 제목으로 설명되어 있으니 참고하시기 바랍니다.

- app.game.resource
  - 게임에서 필요한 이미지와 소리 파일을 관리합니다.
- app.game.common
  - 게임에 필요한 우주선과 같은 기초 클래스들을 관리합니다.

app.game.resource 패키지에 포함된 클래스는 다음과 같습니다.

- Resource: 리소스를 불러 들이기 위해서 Context 객체의 레퍼런스를 보관합니다.
- ResourceSound: 음향 효과 리소스를 관리합니다.
- ResourceBackground: 배경 화면 이미지를 관리합니다.
- ResourceShip: 우주선의 이미지를 관리합니다.
- ResourceLaser: 레이저의 이미지를 관리합니다.
- ResourceAsteroid: 소행성의 이미지를 관리합니다.
- ResourceExplodingAsteroid: 소행성이 폭발했을 때의 이미지를 관리합니다.

app.game.common 패키지에 포함된 클래스는 다음과 같습니다.

- Global: 전역적으로 참조해야 하는 객체에 대한 레퍼런스를 보관합니다.
- Messages: 게임에서 사용하는 메시지의 상수들을 관리합니다.
- GameInformation: 남은 우주선의 개수, 현재 점수 등의 정보를 관리합니다.
- BackgroundImage: 배경 이미지를 표시하고 스크롤 하는 방법을 제공합니다.
- Background: 배경 이미지를 관리합니다. 원근감을 살리기 위하여 배경 이미지 두 장이 서로 다른 속도로 스크롤 하도록 합니다.
- Ship: 우주선을 표시하고 이동합니다.
- ShipAnimater: 우주선이 비행하고 폭발하는 것을 표현합니다.
- Laser: 우주선에서 발사되는 레이저를 표현합니다.
- Lasers: 복수의 레이저 객체를 통합해서 관리합니다.
- Asteroid: 소행성을 표현합니다.
- Asteroids: 복수의 소행성 객체를 통합해서 관리합니다.

2. SceneGame 수정

[소스 1] SceneGame.java

```
  1 : package app.scene;
  2 :
  3 : import ryulib.ValueList;
  4 : import ryulib.game.GameControlGroup;
  5 : import app.game.common.Background;
  6 : import app.game.common.Global;
  7 : import app.game.common.Ship;
  8 : import app.game.level.GameLevel;
  9 : import app.game.level.GameLevel01;
 10 : import app.game.level.GameLevel02;
 11 : import app.game.level.GameLevel03;
 12 :
 13 : public class SceneGame extends Scene {
 14 :
 15 : private static SceneGame _MyObject = null;
 16 :
 17 : public SceneGame(GameControlGroup gameControlGroup) {
 18 :       super(gameControlGroup);
 19 :
 20 :       _MyObject = this;
 21 :
 22 :       gameControlGroup.addControl(this);
 23 : }
 24 :
 25 : public static SceneGame getInstance() {
 26 :    return _MyObject;
 27 : }
 28 :
 29 : private Background _Background = null;
 30 : private Ship _Ship = null;
 31 :
 32 : private GameLevel _GameLevel = null;
 33 :
 34 : public void setGameLevel(int level) {
 35 :       Global.gameInformation.setGameLevel(level);
 36 :
 37 :       if (_GameLevel != null) {
 38 :              _GameLevel.actionOut();
 39 :              _GameLevel = null;
 40 :       }
 41 :
 42 :       switch (level) {
 43 :              case        1:         _GameLevel        =        new
GameLevel01(getGameControlGroup()); break;
 44 :              case        2:         _GameLevel        =        new
GameLevel02(getGameControlGroup()); break;
 45 :              case        3:         _GameLevel        =        new
GameLevel03(getGameControlGroup()); break;
 46 :              default: ; // TODO : raise Exception
 47 :       }
 48 :
 49 :       getGameControlGroup().addControl(_GameLevel);
 50 :
```

```
51 :        _GameLevel.actionIn();
52 : }
53 :
54 : @Override
55 : public void actionIn(Scene oldScene, ValueList params) {
56 :        Global.gameInformation.startGame();
57 :
58 :        _Background = new Background(getGameControlGroup());
59 :        _Ship = new Ship(getGameControlGroup());
60 :
61 :        getGameControlGroup().addControl(_Background);
62 :        getGameControlGroup().addControl(_Ship);
63 :        getGameControlGroup().addControl(Global.gameInformation);
64 :
65 :        setGameLevel(1);
66 : }
67 :
68 : @Override
69 : public void actionOut(Scene newScene) {
70 :        if (_GameLevel != null) {
71 :               _GameLevel.actionOut();
72 :               _GameLevel = null;
73 :        }
74 :
75 :        Global.gameInformation.delete();
76 :        _Ship.delete();
77 :        _Background.delete();
78 :
79 :        _Ship = null;
80 :        _Background = null;
81 : }
82 :
83 : }
```

29-30: 배경 이미지와 우주선을 표현할 객체를 사용할 것입니다.

54-66: 화면이 SceneGame으로 변경되었을 때 실행되는 메소드 입니다.

56: 게임 정보를 관리하는 객체의 레퍼런스는 여러 곳에서 접근할 수 있기 때문에, Global 클래스에 static으로 선언되어 있습니다. 게임이 시작되었음을 알리고 있습니다.

58: 배경 이미지를 관리할 객체를 생성합니다.

59: 우주선을 관리할 객체를 생성합니다.

61-63: 게임관련 객체들을 GameControlGroup에 등록합니다.

65: 게임 레벨을 1부터 시작하도록 합니다. 이번 연재에서는 게임 레벨은 1에서 고정되어 있습니다. 레벨을 클리어하고 다음 레벨로 이동하는 소스는 http://ryulib.tistory.com/ 를 통해서 추후 배포하도록 하겠습니다.

68-81: 게임이 끝나서 SceneGame 화면이 다른 화면으로 전활 될 때 실행되는 메소드 입니다. static으로 선언된 gameInformation을 제외하고는 참조를 null로 지정하여 삭제하고 있습니다. 이는 게임이 메모리를 많이 사용하게 될 경우에, 필요 없는 객체를 바로 바로 삭제하고 필요할 때만 메모리를 사용하도록 하기 위함입니다.

3. GameLevel01 수정

[소스 2] GameLevel01.java

```
  1 : package app.game.level;
  2 :
  3 : import app.game.common.Asteroids;
  4 : import ryulib.game.GameControlGroup;
  5 :
  6 : public class GameLevel01 extends GameLevel {
  7 :
  8 : public GameLevel01(GameControlGroup gameControlGroup) {
  9 :        super(gameControlGroup);
 10 :
 11 : }
 12 :
 13 : private Asteroids _Asteroids = null;
 14 :
 15 : @Override
 16 : public void doActionIn() {
 17 :        _Asteroids = new Asteroids(getGameControlGroup());
 18 :        getGameControlGroup().addControl(_Asteroids);
 19 : }
 20 :
 21 : @Override
 22 : public void doActionOut() {
 23 :        _Asteroids.delete();
 24 :        _Asteroids = null;
 25 : }
 26 :
 27 : }
```

15-19: 해당 게임 레벨이 시작될 때, 실행되는 메소드 입니다.

17-18: 소행성을 관리하는 Asteroids 객체를 생성하고, GameControlGroup에 등록합니다. 이로써, 무작위로 소행성이 생성되어 화면에 스크롤 됩니다. 이때, 우주선이 소행성과 충돌하면 안되며, 레이저를 쏴서 소행성을 폭파시키면 점수가 올라갑니다.

21-25: 해당 게임 레벨이 종료될 때, 실행되는 메소드 입니다.

23-24: SceneGame 때와 마찬가지로 필요 없을 때, 객체를 삭제합니다.

4. SceneResult 수정

[소스 3] SceneResult.java

```
  27 : @Override
  28 : protected void onDraw(GamePlatformInfo platformInfo) {
  29 :        _Paint.setARGB(255, 0, 0, 0);
  30 :        _Canvas.drawRect(0,           0,           _Canvas.getWidth(),
_Canvas.getHeight(), _Paint);
  31 :
  32 :        int point = Global.gameInformation.getPoint();
  33 :
  34 :        _Paint.setARGB(255, 255, 255, 255);
  35 :        _Canvas.drawText("Game Result...", 100, 100, _Paint);
  36 :        _Canvas.drawText("  * Point : " + Integer.toString(point) ,
100, 120, _Paint);
  37 :        _Canvas.drawText("  Touch the screen to continue...", 100,
200, _Paint);
  38 : }
```

우주선이 모두 폭파되면 게임이 끝나면서 SceneResult 화면으로 전환됩니다. 이때 얻은 점수를 표시하는 것만을 추가한 상태입니다.

32: gameInformation 객체에서 현재의 점수를 얻어오고 있습니다.

5. JoyStick 클래스 설명

JoyStick 클래스는 센서나 키보드 또는 터치 이벤트 등을 이용해서 객체의 위치를 변경할 수 있도록 도와주는 클래스입니다. ryulib.game 패키지에 포함되어 있으며, JoyStickinterface 클래스에서 상속받아 확장된 클래스입니다.

JoyStickinterface의 설명

- Method
  - tick(long;
- Property
  - int getX(), int getY(), setX(int), setY(int)
    - 객체의 현재 위치(좌표)가 저장되는 속성입니다.
  - int getSpeed(), setSpeed(int)
    - 객체가 이동하는 속도를 지정합니다. 초당 이동할 수 있는 픽셀 수를 지정합니

다.

- int getLeftLimit(), setLeftLimit(int), int getTopLimit(), setTopLimit(int), int getRightLimit(), set RightLimit(int), int getBottomLimit(), set BottomLimit(int)
  - 객체가 이동할 수 있는 범위가 저장되는 속성입니다. 일반적으로 화면 크기가 지정됩니다.
- int getObjectWidth(), set ObjectWidth(int), int getObjectHeight, setObjectHeight(int)
  - 객체의 크기가 저장되는 속성입니다. 객체가 화면의 오른쪽과 하단을 넘어서지 않도록 합니다.
- boolean isUseBoundaryLimit(), setUseBoundaryLimit(boolean)
  - 객체가 LeftLimit, TopLimit, TopLimit, RightLimit, BottomLimit 범위를 넘어서게 할 것인지를 결정합니다.

6. 우주선 구현

우주선 구현의 기본적인 소스와 설명은 http://ryulib.tistory.com/99 에서 확인하실 수가 있습니다. 구현된 소스에 대한 상세한 설명이 되어 있으며, 다양한 방법을 통해서 우주선이 이동 할 수 있도록 되어 있습니다. 이번 연재에서는 방향 센서를 통해서 우주선을 이동하는 방법을 설명합니다.

[소스 4] Ship.java #1

```
  61 : @Override
  62 : protected void onStart(GamePlatformInfo platformInfo) {
  63 :        _Canvas = platformInfo.getCanvas();
  64 :        _Paint = platformInfo.getPaint();
  65 :
  66 :        getGameControlGroup().addControl(_Lasers);
  67 :
  68 :
       _JoyStick.PrepareOrientationSensor(Resource.getInstance().getContext
());
  69 :
  70 :        _JoyStick.setBoundaryLimit(
  71 :                    true,
  72 :                    0, 0, _Canvas.getWidth(), _Canvas.getHeight(),
  73 :                    ResourceShip.WIDTH, ResourceShip.HEIGHT
  74 :        );
  75 : }
```

68: 방향센서를 이용하기 위해서는 센서를 초기화 해야 합니다.

70: 지정된 범위를 벗어나지 못하도록 객체가 표시되어야 할 공간의 좌표를 지정합니다.

[소스 5] Ship.java #2

```
  77 : @Override
  78 : protected void onTick(GamePlatformInfo platformInfo) {
  79 :        GameMessage msg = platformInfo.getGameMessage();
  80 :        switch (msg.what) {
  81 :               case   Messages.SHIP_EXPLODE:   _ShipAnimater.explode();
break;
  82 :        }
  83 :
  84 :        long tick = platformInfo.getTick();
  85 :        _ShipAnimater.tick(tick);
  86 :        _JoyStick.tick(tick);
  87 :
  88 :        if (Global.gameInformation.getLife() == 0) {
  89 :               _JoyStick.setEnabled(false);
  90 :
  91 :               if (_ShipAnimater.isExploding() == false) {
  92 :                      this.delete();
  93 :                      SceneManager.getInstance().result(null);
  94 :               }
  95 :        }
  96 : }
```

79-82: GameMessage는 게임 컨트롤 객체 간의 메시지 전달의 목적으로 작성된 클래스입니다. 이는 Thread-safe를 보증합니다. 현재 게쪽이 스레드를 사용하고 있고, 게임의 특성상 멀티 스레드를 이용해서 개발해야 할 경우가 많이 생기게 됩니다. 이를 위해서 준비된 클래스입니다. 우선 현재는 수신처리에 대한 설명만 하도록 하겠습니다.

79: 새로운 메시지가 있는 가를 확인합니다. GameMessage는 Broadcast를 기본으로 합니다. 즉, 어느 누군가가 메시지를 발송하면 전체 게임 컨트롤에게 메시지가 전달되는 형식입니다. 자신이 처리해야 할 메시지인지를 알기 위해서는 이미 정해진 규약을 통해서 판단하거나, 메시지 안에 수신자가 식별할 수 있는 데이터를 포함시켜야 합니다. 여기서는 이미 정해진 규약을 통해서 자신이 사용할 메시지를 안다고 가정하고 있습니다. 이를 통해서 얻을 수 있는 이점은 메시지를 발송하는 쪽이나 수신하는 쪽 모두 서로를 의식하지 않고 개발할 수 있다는 점 입니다. 결국 결합도가 낮아져서 유연한 코드를 유지할 수 있도록 합니다.

80: GameMessage에는 여러 가지 정보를 담을 수 있도록 되어 있습니다. 여기서는 waht이라는 정수형 변수를 통해서 어떤 종류의 메시지인지를 판단하도록 되어 있습니다.

81: Messages 클래스에 정의된 SHIP_EXPLODE 메시지가 전달되면 자신의 우주선을 폭발하도록 합니다. ShipAnimater 객체는 우주선의 애니메이션 동작을 컨트롤하는 기능을 담당합니다.

85: 우주선의 현재 애니메이션을 tick(int) 메소드에 전달된 시간을 기준으로 표현합니다. 현재 우주선은 비행하는 동작과 폭발하는 동작 두 가지로 나누어져 있습니다. 폭발의 경우에는 실제 폭발은 아니고, 우주선이 흔들리는 것으로 표현되고 있습니다.

86: JoyStick 객체의 경우에도 tick(int) 메소드를 통해서 전달된 시간을 기준으로 우주선을 이동하고 있습니다. 이는 스마트 폰의 성능에 따라 다른 속도로 게임이 진행되는 것을 방지합니다.

88-95: 우주선이 모두 폭발하고 남은 것이 없다면, JoyStick을 Disable 시키고, 우주선 객체를 삭제합니다.

91-93: 우주선이 폭발 중 애니메이션 동작을 하는 동안에는 우주선 객체를 삭제하지 않고 기다립니다. 우주선이 폭발 애니메이션이 다 끝난 경우에는 게임 결과 화면으로 전환합니다.

[소스 6] Ship.java #3

```
  98 : @Override
  99 : protected void onDraw(GamePlatformInfo platformInfo) {
 100 :                _Canvas.drawBitmap(
 101 :
       ResourceShip.getInstance().getBitmap(_ShipAnimater.getIndex()),
 102 :                              _JoyStick.getX(), _JoyStick.getY(),
 103 :                              _Paint
 104 :                );
 105 :
 106 :                GameControl GameControl = this.checkCollision(this);
 107 :                if  ((GameControl  !=  null)  &&  (GameControl  instanceof
Asteroid)) {
 108 :                       explode();
 109 :                       ((Asteroid) GameControl).explode();
 110 :                }
 111 :        }
```

106-110: 우주선과 충돌하는 객체를 찾아서, 이것이 소행성일 경우에는 우주선과 소행성 모두를 폭발 시키는 과정입니다. 추후에는 게임 레벨이 높아지면서 소행성 대신 다른 객체들을 등장시킬 예정이기 때문에 이 부분은 수정이 가해져야 하는 부분입니다.

[소스 7] Ship.java #4

```
 113 :        @Override
 114  :      protected  boolean  onTouchEvent(GamePlatformInfo  platformInfo,
MotionEvent event) {
 115 :                _Lasers.Fire(
 116 :                              _JoyStick.getX() + ResourceShip.WIDTH,
 117 :                              _JoyStick.getY()  +  (ResourceShip.HEIGHT  /
2));
 118 :
 119 :                return true;
 120 :     }
 121 :
 122 : }
```

115-117: 화면 터치가 일어나면 레이저를 발사하도록 합니다. 발사되는 위치는 우주선의 전방 중심입니다.

7. 소행성 구현

소행성에 대한 기본적인 설명은 http://ryulib.tistory.com/102 을 참고하시기 바랍니다.

[소스 8] Asteroid.java

```
  69 : public void explode() {
  70 :        Global.gamePlatform.getGamePlatformInfo().addMessage(this,
Messages.ASTEROID_EXPLODE);
  71 :
  72 :        _AnimationCounter.clear();
  73 :
       _AnimationCounter.setSize(ResourceExplodingAsteroid.getInstance().ge
tCount());
  74 :        _AnimationCounter.setAutoRewind(false);
  75 :
  76 :        _BitmapList = ResourceExplodingAsteroid.getInstance();
  77 :
  78 :        ResourceSound.getInstance().boom();
  79 : }
```

70: GameMessage를 발송하는 과정입니다. gamePlatformInfo 객체의 addMessage() 메소드는 여러 수준의 파라메터를 지원하도록 overload 되어 있습니다. 여기서는 정수형 메시지만을 전송하는 경우입니다.

72-78: 소행성의 폭발 애니메이션을 처리하고 있습니다. 우주선과 다르게 두 종류의 BitmapList를 통해서 일반 애니메이션과 폭발 애니메이션을 구별하도록 되어 있습니다. 우주선의 경우에는 모든 이미지가 하나의 BitmapList를 이루고 있고, 몇 번째부터 몇 번째 이미지까지 사용할 지를 결정하여 구별하도록 되어 있습니다. 우주선의 경우에는 ShipAnimater 클래스가 애니메이션을 전담하도록 되어 있습니다. 필자의 경우에는 우주선과 같은 처리를 선호합니다.

79: 소행성이 폭발하는 소리를 냅니다.

8. 음향효과 입히기

이번 연재에서 사용하는 소스에서는 음향효과를 위해서 SoundPool을 사용하고 있습니다. 안드로이드에 포함된 JetBoy의 경우에는 JetPlayer를 사용하고 있습니다.

[소스 9] ResourceSound.java

```
   1 : package app.game.resource;
   2 :
   3 : import android.media.AudioManager;
```

```
  4 : import android.media.SoundPool;
  5 : import app.main.R;
  6 :
  7 : public class ResourceSound {
  8 :
  9 : // 최대 스트림 갯수
 10 : private static final int _STREAM_COUNT = 4;
 11 :
 12 : private static ResourceSound _Instance = new ResourceSound();
 13 :
 14 : public static ResourceSound getInstance(){
 15 :        return _Instance;
 16 : }
 17 :
 18 : private ResourceSound() {}
 19 :
 20 : private  SoundPool  _SoundPool    =  new  SoundPool(_STREAM_COUNT,
AudioManager.STREAM_MUSIC, 0);
 21 :
 22 : private int _Fire = -1;
 23 : private int  Boom = -1;
 24 :
 25 : public void prepare() {
 26 :        _Fire = _SoundPool.load(
 27 :                    Resource.getInstance().getContext(),
 28 :                    R.raw.fire,
 29 :                    1
 30 :              );
 31 :
 32 :        _Boom = _SoundPool.load(
 33 :                    Resource.getInstance().getContext(),
 34 :                    R.raw.boom,
 35 :                    1
 36 :              );
 37 : }
 38 :
 39 : public void fire() {
 40 :        _SoundPool.play(_Fire, 1, 1, 0, 0, 1);
 41 : }
 42 :
 43 : public void boom() {
 44 :        _SoundPool.play(_Boom, 1, 1, 0, 0, 1);
 45 : }
 46 :
 47 : }
```

20: SoundPool 객체를 생성합니다. 파라메터는 최대 스트림 개수, 스트림 타입, 그리고 품질로 나누어져 있습니다. 이중에서 품질은 아직 사용되지 않는 것으로 0으로 넘겨주면 됩니다.

25-37: 음원 데이터를 처음 로딩할 때 시간이 다소 걸릴 수가 있습니다. 따라서 사용하기 전에 미리 로딩해서 사용합니다.

39-41: 레이저 발사하는 음향효과를 냅니다.

43-45: 소행성이 폭발하는 음향효과를 냅니다.

9. GameInformation 구현

[소스 10] GameInformation.java

```
  1 : package app.game.common;
  2 :
  3 : import ryulib.game.GameControl;
  4 : import ryulib.game.GameControlGroup;
  5 : import ryulib.game.GameMessage;
  6 : import ryulib.game.GamePlatformInfo;
  7 : import android.graphics.Canvas;
  8 : import android.graphics.Paint;
  9 :
 10 : public class GameInformation extends GameControl {
 11 :
 12 : public static final int SHIP_COUNT = 3;
 13 :
 14 : public GameInformation(GameControlGroup gameControlGroup) {
 15 :        super(gameControlGroup);
 16 :        // TODO Auto-generated constructor stub
 17 : }
 18 :
 19 : private Canvas _Canvas = null;
 20 : private Paint _Paint = null;
 21 :
 22 : private int _GameLevel = 0;
 23 : private int _Life = 0;
 24 : private int _Point = 0;
 25 :
 26 : public void startGame() {
 27 :        setGameLevel(0);
 28 :        _Life = SHIP_COUNT;
 29 :        _Point = 0;
 30 :
 31 :        this.setEnabled(true);
 32 :        this.setVisible(true);
 33 : }
 34 :
 35 : @Override
 36 : protected void onStart(GamePlatformInfo platformInfo) {
 37 :        _Canvas = platformInfo.getCanvas();
 38 :        _Paint = platformInfo.getPaint();
 39 : }
 40 :
 41 : @Override
 42 : protected void onDraw(GamePlatformInfo platformInfo) {
 43 :        GameMessage msg = platformInfo.getGameMessage();
 44 :        switch (msg.what) {
 45 :              // TODO : GameLevel 마다 다른 점수 부여
 46 :              case Messages.ASTEROID_EXPLODE: _Point += 100; break;
 47 :
 48 :              case  Messages.SHIP_EXPLODE:  if  (_Life  >  0)  _Life--;
```

```
break;
  49 :        }
  50 :
  51 :        String   info   =   String.format("Level:   %2d,   Life:   %2d,
Point: %6d", _GameLevel, _Life, _Point);
  52 :
  53 :        _Paint.setARGB(255, 255, 255, 255);
  54 :        _Canvas.drawText(info, 10, 20, _Paint);
  55 : }
  56 :
  57 : public void setGameLevel(int value) {
  58 :        _GameLevel = value;
  59 : }
  60 :
  61 : public int getGameLevel() {
  62 :        return _GameLevel;
  63 : }
  64 :
  65 : public int getLife() {
  66 :        return _Life;
  67 : }
  68 :
  69 : public int getPoint() {
  70 :        return _Point;
  71 : }
  72 :
  73 : }
```

GameInformation 클래스는 게임에 대한 정보를 관리합니다.

26-33: 게임이 시작되었을 때, 모든 데이터를 초기화 합니다.

41-55: 현재의 정보를 화면에 표시합니다. 현재는 게임 레벨, 남은 우주선 개수, 점수를 문자로 표시하고 있습니다.

43-49: GameMessage 객체를 통해서 점수를 올리거나, 남은 우주선 개수를 줄여 나갑니다.