안드로이드 개발 - 5회

게임으로 배우는 안드로이드 개발

이제 연재의 마무리를 하게 되었습니다. 이번 연재에서는 지난 연재에서 구축한 슈팅 게임의 플랫폼에 실제 게임 모듈을 얹어서 슈팅게임을 완성하도록 하겠습니다. 이번 연재에서는 소리를 다루는 방법과 방향 센서를 이용하여 우주선을 움직이는 방법을 다뤄보도록 하겠습니다.

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

연재순서

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

1. 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 전환 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 지정합니다.
  - 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.

10. 연재를 마치며

우선, 지금까지 부족한 연재를 지켜봐 주신 독자들에게 감사 합니다. 연재를 시작하기 이전부터 개인적으로 많은 시간을 투자해서 준비했던 내용이지만, 엔진 자체도 안정화가 안된 상태에서 시작했던 만큼 아쉬움이 많이 남습니다. 이후부터는 제 블로그(http://ryulib.tistory.com/)를 통해서 게쪽은 지속적으로 업그레이드 될 예정입니다. 아울러 게임의 종류도 점점 더 추가할 것이니 관심 있는 분들은 자주 방문해주시면 감사하겠습니다.