군산대학교
KUNSAN NATIONAL UNIVERSITY

SensorSimulator를 이용한 게임 구현 및 과제

모바일 응용

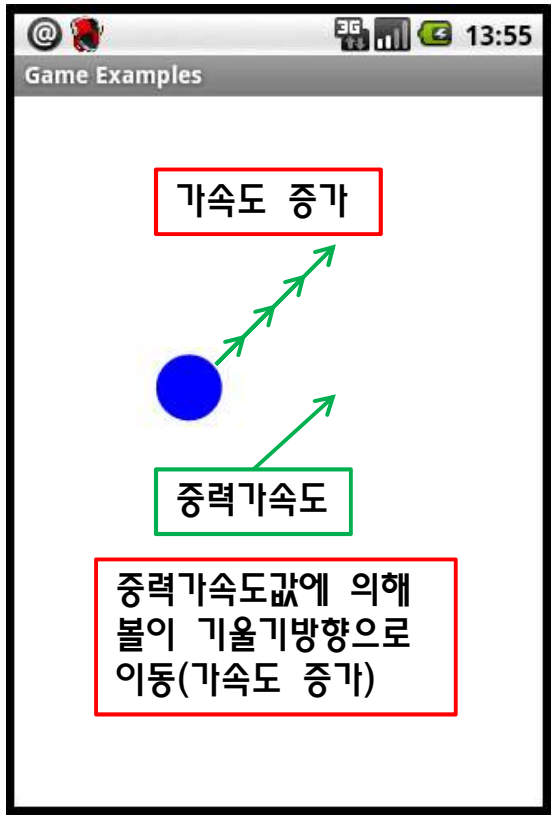
남광우

SensorSimulator를 이용한 게임 구현

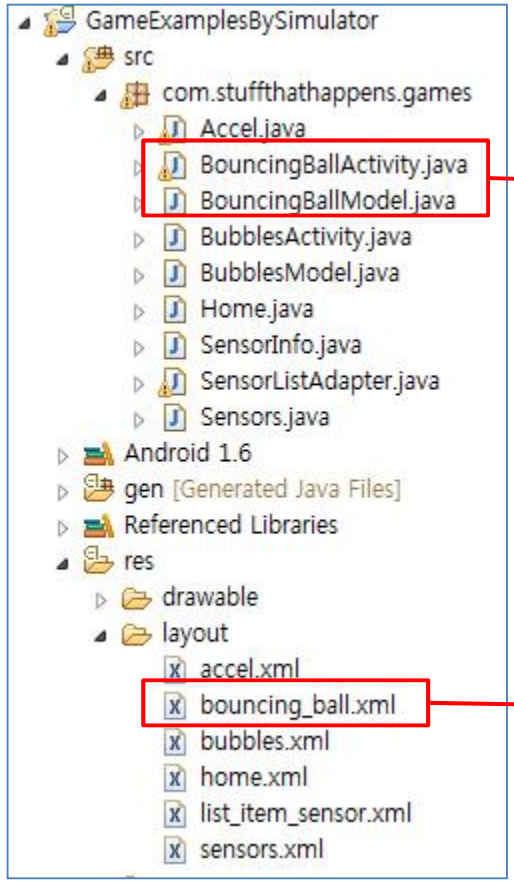
❖ Bouncing Ball 게임

http://stuffthathappens.com/blog/2009/03/21/android-bouncing-ball-demo/

바운싱볼 게임의 화면



바운싱볼 게임의 주요 소스 파일



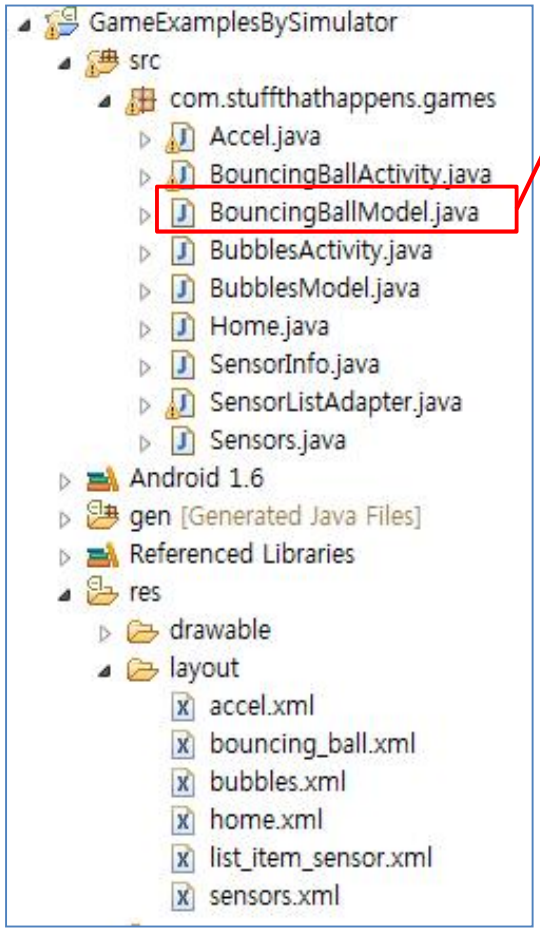
바운싱볼 객체와 바운싱볼 게임 액티비티에 관련된 Java 파일

시작시 메인화면 액티비티

바운싱볼 액티비티에 관한 XML 파일

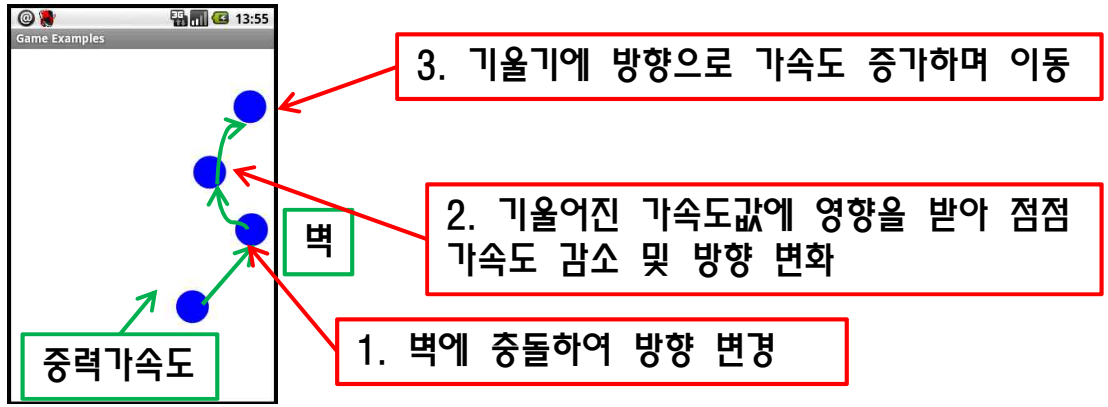
SensorSimulator를 이용한 게임 구현

❖ BouncingBallModel.java



바운싱볼 객체에 대한 파일로 객체의 위치 및 상태 데이터와 벽에 부딛쳤을때 리바운딩되는 물리적인 연산에 대한 처리를 포함하고 있음. BouncingBallActivity에서 객체로 선언하여 사용.

물리적인 연산에 대한 것들이 많아 다음과 같이 간단히 설명하고 넘어감. 볼의 움직임은 센서로 부터 넘어오는 중력가속도값을 통해 볼이 현재 중력가속도 값과 계산하여 힘을 받는 쪽으로 X, Y값을 변화하여 볼을 이동시키는 원리로 되어있습니다. 그리고 화면의 높이와 넓이 값을 지정하여 볼이 화면 벽에 다았을경우 충동한 면의 반대 방향으로 중력가속도 값을 변환하여 볼이 반대 방향으로 튕겨나가는 연산을 객체 내부에서 처리하도록 구성되어 있음.



SensorSimulator를 이용한 게임 구현

❖ BouncingBalActiviity.java(1)

```
package com.stuffthathappens.games;

import org.openintents.sensorsimulator.hardware.SensorManagerSimulator;

import static android.hardware.SensorManager.DATA_X;
import static android.hardware.SensorManager.DATA_Y;
import static android.hardware.SensorManager.SENSOR_ACCELEROMETER;
import static android.hardware.SensorManager.SENSOR_DELAY_GAME;

import java.util.concurrent.TimeUnit;

import android.app.Activity;
import android.app.AlertDialog;
import android.app.Dialog;
import android.content.DialogInterface;
import android.graphics.Canvas;
import android.graphics.Color;
import android.graphics.Paint;
import android.hardware.SensorListener;
import android.hardware.SensorManager;
import android.os.Bundle;
import android.os.Vibrator;
import android.view.SurfaceHolder;
import android.view.SurfaceView;
import android.view.SurfaceHolder.Callback;
```

기본센서가 아닌 센서 시뮬레이터를 이용하기 위하여 추가한 라이브러리를 임포트

BouncingBallActivity를 위한 기본 라이브러리들

SensorSimulator를 이용한 게임 구현

❖ BouncingBalActiviity.java(2)

```
public class BouncingBallActivity extends Activity implements Callback, SensorListener {
    private static final int BALL_RADIUS = 20;
    private SurfaceView surface;
    private SurfaceHolder holder;
    private final BouncingBallModel model = new BouncingBallModel(BALL_RADIUS);
//  private final BouncingBallModel model1 = new BouncingBallModel(BALL_RADIUS);
    private GameLoop gameLoop;
    private Paint backgroundPaint;
    private Paint endBox;
    private Paint ballPaint;
    private SensorManagerSimulator sensorMgr;
    private long lastSensorUpdate = -1;

//  AlertDialog.Builder ad;

    int box_top_x = 100;
    int box_top_y = 50;
    int box_bottom_x = 180;
    int box_bottom_y = 130;
```

게임에 사용될 공(Ball)의 반지름

게임화면을 담당할 SurfaceView 객체

SurfaceView 객체의 화면을 직접적으로 접근하여 화면을 다루는 객체

BouncingBallModel객체 생성과 함께 볼의 반지름 지정)

백그라운드 화면과 볼, 그리고 과제에서 필요한 상자에 대한 페인트객체 생성

센서시뮬레이터 객체 생성

생성할 상자(Box)의 크기를 위한 변수 선언 및 좌표지정
(값은 상자의 길이 값이 아닌 top, Bottom에 대한 화면상의 좌표값)

SensorSimulator를 이용한 게임 구현

❖ BouncingBalActiviity.java(3)

```
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

    setContentView(R.layout.bouncing_ball);

    surface = (SurfaceView) findViewById(R.id.bouncing_ball_surface);
    holder = surface.getHolder();
    surface.getHolder().addCallback(this);

    backgroundPaint = new Paint();
    backgroundPaint.setColor(Color.WHITE);
    endBox = new Paint();
    endBox.setColor(Color.RED);

    ballPaint = new Paint();
    ballPaint.setColor(Color.BLUE);
    ballPaint.setAntiAlias(true);
```

리소스의 Bouncing_Ball.xml에 대한 리소스 추가

Surface객체에 리소스의 SurfaceView지정

Holder객체를 통해 surface객체를 접근하여 화면처리를 위한 코드

백그라운드 화면을 위한 객체 생성 및 색 지정

상자(Box)화면을 위한 객체 생성 및 색 지정

볼 화면을 위한 객체 생성 및 색 지정
(setAntiAlias는 부드러운 곡선처리를 위한 옵션)

SensorSimulator를 이용한 게임 구현

❖ BouncingBalActiviity.java(4)

```
@Override
protected void onPause() {
    super.onPause();

    model.setVibrator(null);
    sensorMgr.unregisterListener(this, SENSOR_ACCELEROMETER);
    sensorMgr = null;
    model.setAccel(0, 0);
}

@Override
protected void onResume() {
    super.onResume();
    sensorMgr = SensorManagerSimulator.getSystemService(this, SENSOR_SERVICE);
    sensorMgr.connectSimulator();
    boolean accelSupported = sensorMgr.registerListener(this,
            SENSOR_ACCELEROMETER,
            SENSOR_DELAY_GAME);
    if (!accelSupported) {
        // on accelerometer on this device
        sensorMgr.unregisterListener(this, SENSOR_ACCELEROMETER);
        // TODO show an error
    }
    // NOTE 1: you cannot get system services before onCreate()
    // NOTE 2: AndroidManifest.xml must contain this line:
    // <uses-permission android:name="android.permission.VIBRATE"/>
    Vibrator vibrator = (Vibrator) getSystemService(Activity.VIBRATOR_SERVICE);
    model.setVibrator(vibrator);
}
```

정지시 관련 객체에 대한 초기화 및 설정해제

기본소스상에 없는 부분(프로그램 구동시 센서시뮬레이터가 잘 작동하지 않는 경우 이 부분을 추가하면 시뮬레이터 잘 작동)

재동작시 관련 객체에 대한 초기화 및 설정

디바이스(스마트폰)의 진동센서 생성 및 볼에 진동기능 설정

SensorSimulator를 이용한 게임 구현

❖ BouncingBalActiviity.java(5)

```
public void surfaceChanged(SurfaceHolder holder, int format, int width,
        int height) {

    model.setSize(width, height);
    model1.setSize(width, height);
}

public void surfaceCreated(SurfaceHolder holder) {
    gameLoop = new GameLoop();
    gameLoop.start();
}

private void draw() {
    // TODO thread safety - the SurfaceView could go away while we are drawing

    Canvas c = null;
    try {
        // NOTE: in the LunarLander they don't have any synchronization here,
        // so I guess this is OK. It will return null if the holder is not ready
        c = holder.lockCanvas();

        // TODO this needs to synchronize on something
        if (c != null) {
            doDraw(c);
        }
    } finally {
        if (c != null) {
            holder.unlockCanvasAndPost(c);
        }
    }
}
```

서페이스에 상태가 변화되었을 때 자동으로 동작되는 함수

서페이스가 생성과 동시에 자동적으로 동작하는 함수로 내부적으로 GameLoop(쓰레드를 상속받은) 객체를 생성하고 작동함

Surfaceholder를 잠금

그려질 객체의 동기화 및 객체 그리기

Surfaceholder를 잠금해제

SensorSimulator를 이용한 게임 구현

❖ BouncingBalActiviity.java(6)

```
private void doDraw(Canvas c) {
    int width = c.getWidth();
    int height = c.getHeight();
    c.drawRect(0, 0, width, height, backgroundPaint);
```

백그라운드에 대한 길이와 높이에 대한 설정 및 백그라운드 그리기

```
    float ballX, ballY;
    synchronized (model.LOCK) {
        ballX = model.ballPixelX;
        ballY = model.ballPixelY;
```

모델의 객체로서 데이터를 동기화함

```
    }
    c.drawRect(box_top_x, box_top_y, box_bottom_x, box_bottom_y, endBox);
    c.drawCircle(ballX, ballY, BALL_RADIUS, ballPaint);
}
```

상자(box)와 봉(ball) 그리기

```
public void surfaceDestroyed(SurfaceHolder holder) {
    try {
        model.setSize(0,0);
        gameLoop.safeStop();
    } finally {
        gameLoop = null;
    }
}
```

Surface객체가 소멸될 때 같이 소멸될 객체나 변수 해제

SensorSimulator를 이용한 게임 구현

❖ BouncingBalActiviity.java(7)

```
private class GameLoop extends Thread {
    private volatile boolean running = true;

    public void run() {
        while (running) {
            try {
                // TODO don't like this hardcoding
                TimeUnit.MILLISECONDS.sleep(5);

                draw();
                model.updatePhysics();
                ballCollision();
                model1.updatePhysics();

            } catch (InterruptedException ie) {
                running = false;
            }
        }
    }

    public void safeStop() {
        running = false;
        interrupt();
    }
}
```

게임실행시 수행될 쓰레드

Draw 함수로 객체를 그리고 모델 객체의 물리적인 처리 수행 후 ballCollision이라는 상자와 볼의 충돌체크를 위한 함수 수행

쓰레드 정지시 인터럽트로 쓰레드 정지

SensorSimulator를 이용한 게임 구현

❖ BouncingBalActiviity.java(8)

```
public void onAccuracyChanged(int sensor, int accuracy) {
}

public void onSensorChanged(int sensor, float[] values) {
    if (sensor == SENSOR_ACCELEROMETER) {
        long curTime = System.currentTimeMillis();
        // only allow one update every 50ms, otherwise updates
        // come way too fast
        if (lastSensorUpdate == -1 || (curTime - lastSensorUpdate) > 50) {
            lastSensorUpdate = curTime;

            model.setAccel(values[DATA_X], values[DATA_Y]);
        }
    }
}

public void ballCollision()
{
    if(box_top_x <= model.ballPixelX-20 && box_top_y <= model.ballPixelY-20 &&
            box_bottom_x >= model.ballPixelX+20 && box_bottom_y >= model.ballPixelY+20)
    {

        ?
        충돌체크(상자안에 공이 들어갔을때)시 사용자가 원하는 행동 코딩

    }
}
```

센서값이 변했을때 자동적으로 실행되는 함수

일정시간(50/1000초단위로) 모델(ball) 객체의 중력가속도 값 갱신

상자(Box)와 모델(Ball)간의 충돌 체크를 위한 함수

상자(Box)와 모델(Ball)간의 위치가 비교

SensorSimulator를 이용한 게임 구현

❖ 실습 과제

- ✓ 3~5개의 색이 다른 공을 만들어 상자 안에 들어갈 때 마다 스코어를 계산하여 마지막 공까지 상자에 들어갈 경우 게임으로 종료하게 하시오