**- 안드로이드 개발에 사용되는 언어**

Java, Kotlin(코틀린) : 둘 다 같이 사용가능.

**- 프로젝트 시작 시 입력하는 것들 중 bundle ID (package name)의 구성 방법**

Organization name(회사) + application name(앱) : com.회사 이름(도메인).프로그램 이름

**- 프로젝트 시작 시 생성되는 파일들에 대하여**

1. MainActivity.java : 앱을 실행하면 MainActivity의 인스턴스를 실행하고 레이아웃을 로드.

2. activity\_main.xml : UI 레이아웃 정의.

3. AndroidManifest.xm(매니페스트 파일) : 앱의 기본 특성을 설명하고 앱의 각 구성요소를 정의.

4. Res : 리소스 파일 폴더.

5. build.gradle: 프로젝트 전체에 공통으로 적용할 설정 기술.

\_ build script : 외부 저장소(repositories)와 의존성(dependencies) 설정

**- Android Studio & Gradle build system**

Android Studio와 build system은 서로 독립적.

Android Studio는 코드 편집만 담당하고, build는 Gradle을 통해 수행.

**- Android API Level, Version Number, Code name 등**

1. API Level : Android 플랫폼 버전과 애플리케이션 호환성을 API 수준 정수로 표시.

2. Version Number : 7.0, 7.1 등 우리가 흔히 아는 버전 표시.

3. Code name : Version을 디저트 이름을 사용해서 표시.

**- 실기기를 개발용으로 전환하는 방법**

1. 기기에서 소프트웨어 정보 – 빌드 번호 7번 터치하여 개발자 모드로 변경.

2.개발자 옵션에서 USB 디버깅 설정 후 연결.

**- Bitmap Resource 사용법 및 다양한 해상도에 대해 대비하기**

Resources res = getResources();

Bitmap bitmap = BitmapFactory.decodeResource(res, R.mipmap.file\_name); // Resource폴더에 저장된 파일 Bitmap으로 리턴

Rect srcRect = new Rect();

srcRect.set(0, 0, bitmap.getWidth(), bitmap.getHight());

Rect dstRect = new Rect();

dstRect.set(0, 0, 100, 100); // (0, 0)에 100x100 크기로 그림.

Canvas.drawBitmap(bitmap, srcRect, dstRect, null);

해상도, 언어, 기종 별로 이미지를 준비

**- String Resource 사용법 및 다양한 언어에 대해 대비하기**

[ string.xml ]

<resources>

<string name = “app\_name”>카드게임</string>

<string name = “Restart”>재시작<</string>

</resources>

“ “ 대신 R.string.string\_name 사용.

1. i18n(internationalization, 국제화) : 프로그램을 지역에 맞게 조정하는 것. 리소스를 빼는 작업.

2. L10n(localization) : 영어로 되어 있는 프로그램을 한국어로 지원하게 하는 것.

**- Resource Filename 사용규칙**

Java의 이름 작성 규칙을 따름

1. 소문자만 사용 가능 2. 특수문자는 \_와 $만 사용 가능 3. 숫자 사용 불가

**- Layout**

1. FrameLayout : 여러 개의 뷰를 중첨으로 배치하고 그 중 하나를 레이아웃 전면에 표시할 때 사용하는 레이아웃.

같은 크기의 뷰를 여러 개 배치할 경우 가장 나중에 배치된 뷰만 보임.

크기가 다를 경우 뒤에 배치된 뷰의 일부가 보일 수 있음.

2. LinearLayout : 뷰를 수평 또는 수직 방향으로 배치할 수 있는 레이아웃.

3. RelativeLayout : 부모 뷰 또는 자식 뷰의 상대적 위치관계를 정의해 배치하는 레이아웃.

4. ConstraintLayout : 레이아웃 구성 시 뷰 위젯의 위치와 크기를 유연하게 조절할 수 있게 만들어주는 레이아웃.

**- Layout.xml 속성**

1. layout\_width : 뷰의 너비 설정.

2. layout\_height : 뷰의 높이 설정.

3. layout\_weight : 뷰의 가중치를 설정.

4. layout\_gravity : 부모 뷰 내에서 위치를 설정.

**- Units(단위)**

1. px : 화면을 구성하는 최소단위인 픽셀 수.

2. dp : 밀도 독립 픽셀 단위. 화면 해상도에 상관없이 동일한 크기를 갖음.

3. dpi : 1인치에 들어가는 픽셀을 나타내는 단위.

4. sp : 스케일 독립 픽셀 단위. 글꼴의 단위. 사용자 설정에 따라 바뀜.

5. in : 화면의 물리적 사이즈 기반으로 인치를 사용.

6. mm : 화면의 물리적 사이즈 기반으로 밀리미터 사용.

7. em : 글꼴과 상관없이 동일한 텍스트의 크기를 출력.

**- View ID 및 Runtime에 이 값들을 활용하기**

소스 코드 상에서 해당 컨트롤을 접근하기 위해 사용

**- findViewById의 비용**

findViewById() : id가 0이상인 경우 findViewTraversal 호출.

findViewTraversal() : id가 view의 id가 같은 지 검사하고 아닐 경우 null 반환.

ViewGroup의 id와 같은 지 확인 후 같으면 자기자신 반환, 아니면 자기자신이 가진 ChildView에 대해 loop를 돌려 findViewById 호출.

비용이 많이 들게 됨 -> 생성 시점에서 한번만 사용하는 것이 좋음

**- XML의 기본 구조, 구성 요소. Elements/Tags, Attributes, DTD, Schema**

< : Elements/Tags.

Android::textSize 등 : Attributes.

DTD : xml문서를 정의하는 문서.

Schema : DTD를 다시 xml로 정의한 거.

XML : <tag/> 형식 사용.

<shop/>과 같이 단일로 사용하기도 하고, <shop></shop>과 같이 시작과 종류 트리를 사용하기도 함.

이때 shop은 태그명으로 각 데이터의 항목명이 됨.

<shop>과 </shop> 사이에는 순수 데이터 값 또는 다른 태그들이 중복으로 들어갈 수 있음.

장점 : 1. 주석 사용 가능 2. 요구된 구조화 형태를 제대로 갖췄는지 스키마를 통해 확인 가능

**- JSON의 기본 구조, 구성 요소. Key/Value. Boolean/Number/String/Array/Object**

key : value 구조.

{

객체(Object)

key : 문자열 사용(“ “)

value : 기본 자료형, 배열(Array)[], 객체(Object)key/value쌍들의 비순서화된 set

}

JSON : 자바 스크립트의 객체 표기법.

XML보다 좀 더 간결하며 {}으로 항목과 데이터를 구분.

객체 안에는 단일 데이터 또는 또 다른 데이터가 들어가고 []을 사용해 배열을 사용할 수 있음.

단점 : 문법 오류에 취약함.

**- View/Component/Widget properties: TextView, Button, ImageView, EditText, CheckBox, Slider, ...**

1. TextView : 글씨

android:text : 내용 설정.

android:textSize : 텍스트 크기 설정.

android:textColor : 텍스트 색 설정.

android:textStyle : 글꼴 설정.

andorid:autoLink : 설정된 링크 열어줌.

android:autoSizeTextType : TextView 크기에 맞게 텍스트 크기 자동으로 설정.

android:ellipsize : 텍스트 ..으로 생략해서 표시.

2. ImageView : 이미지

android:src : 이미지 설정.(이미지 크기만큼만 나옴)

android:background : 이미지 설정.(ImageView 전체를 덮음)

android:scaleType : 이미지 출력 크기 및 비율 설정.

3. EditText : 글씨 입력창

android:lines : 입력창이 차지하는 줄 수 설정.(처음부터 설정한 줄 수가 나타남)

android:maxLines : 입력창이 차지하는 줄 수 설정.(입력할 때마다 줄 수 늘어남)

android:inputType : 입력값 종류 설정.

4. Button : 버튼

android:text : 텍스트 설정.

android:drawableLeft : 텍스트와 아이콘 같이 사용하는 경우 사용.

android:visibility : 보이게 할 것인지 설정.

androdi:enabled : 활성화할 것인지 설정.

5. CheckBox : 체크상자

android:visibility : 보이게 할 것인지 설정.

androdi:enabled : 활성화할 것인지 설정.

6. RadioButton : 세트에서 한 가지만 선택 가능

android:checked : 버튼 체크 상태 설정.

android:text : 출력할 텍스트 설정.

7. Switch : on/off 버튼

android:showText : on/off 텍스트가 보이게 할 것인지 설정.

android:thumb : 사용자 드래그를 통해 on/off 설정 가능하도록 설정.

8. ToggleButton : on/off 토글 버튼

android:textOff : toggle이 off인 상태일 때 버튼에 나타날 텍스트 설정.

android:textOn : toggle이 on인 상태일 때 버튼에 나타날 텍스트 설정.

9. Spinner : 클릭시 아래 메뉴 나옴

android:dropDownWidht : 스피너를 클릭했을 때 열리는 메뉴의 가로폭 설정.

android:spinnerMode : 스피터 형태 설정.

10. ListView

android:divider : 아이템 간 구분을 위한 색상, 이미지 설정.

android:divider : divider 높이 지정.

android:fastScrollEnabled : 스크롤 할 때 thumb 사용할 것인지 설정.

11. SeekBar : 음량 조절 같은 바

android:max : SeekBar의 최대값 설정.

android:thumb : 커서 이미지 설정.

**- Log Levels**

Logcat에 표시할 메시지 수 제어

1. Verbose : 모든 로그 메시지 표시.(기본 설정)

2. Debug : 개발 단계에서만 유용한 디버그 로그 메시지 뿐 아니라 그보다 낮은 레벨의 메시지 표시.

3. Info : 일반적인 사용을 위해 예상할 수 있는 로그 메시지 뿐 아니라 그보다 낮은 레벨의 메시지 표시.

4. Warn : 아직 오류는 아니지만 발생할 수 있는 문제 뿐 아니라 그보다 낮은 레벨의 메시지도 이 목록에 표시.

5. Error : 오류를 일으킨 문제 뿐 아니라 그보다 낮은 레벨의 메시지도 이 목록에 표시.

6. Assert : 개발자가 발생해서는 안 된다고 생각하는 문제를 표시.

**- View.tag**

Tag : 꼬리표, 번호표

xx.setTag(“tag”); // 태그 설정.

xx.getTag(); // 태그 가져오기.

**- Padding vs Margin**

Padding : 현재 위젯 테두리부터 안의 내용 사이에 여백을 두는 것.

Margin : 부모 레이아웃 테두리부터 위젯 사이에 여백을 두는 것.

**- Activity Restart on Configuration Change**

Configuration Change : 언어 변경, 가로세로 변경 등. 변경되더라도 상태를 유지할 설정값을 지정.

<?xml version="1.0" encoding="utf-8"?>

<manifest xmlns:android="http://schemas.android.com/apk/res/android"

package="com.example.cardsapp">

<application

…

<activity

…

android:configChanges="orientation|screenSize"

// orientation : 가로, 세로 모드 전환 시 설정 유지.(screenSize도 같이 설정해야함)

android:screenOrientation="portrait"> // screenOrientation=”portrait” : 세로, “landscape” : 가로

**- Garbage Collection in Java**

할당된 메모리를 개발자가 해제해주지 않아도 알아서 정리해주는 것.

**- Canvas save()/restore(), Transform matrix**

canvas.save() : 저장(push)

transform matrix : 이동 및 회전. 3x3 행렬을 2차원으로 변환. 모든 뷰가 행렬을 하나 가지고 있음.

draw;

canvas.restore() : 사용하기 전 환경 저장(pop)

스택 구조(Last In First Out)

**- Event-driven programming**

Event-driven : 프로그램 실행 주도권을 시스템이 가지는 것 <-> 프로그램 실행 주도권을 코드에서 가지는 것

**- Android Studio Tool 사용법**

1. Alt+Enter: Import, Error correction, ...

2. Refactoring: Extract/Introduce, ...

- Shift + F6 : 이름 변경

- Ctrl + Alt + M : 메서드 추출

- Ctrl + Alt + V : 변수 추출

- Ctrl + Alt + F : 필드 추출

- Ctrl + Alt + C : 상수 추출

- Ctrl + Alt + P : 매개변수 추출

- Pull up : 하위 클래스의 함수를 상위 클래스로 올리는 것 반대) pull down

3.

- Alt + 1 : 프로젝트 도구 창 표시 여부 전환

- Shift + ESC : 활성 또는 마지막 활성 도구 창 숨기기

- Shift + Ctrl + Backspace

- Ctrl + / : 줄 주석으로 주석 처리, 주석 처리 제거

- Ctrl + Shift + / : 블록 주석으로 주석 처리, 주석 처리 제거

**- Thread Safety with Android**

Thread Safety는 꼭 필요한 곳에서만 사용

Android는 Thread Safety x

**- Design Patterns: Builder, Singleton, Factory, ...**

1. Builder : 객체의 생성 단계들을 캡슐화해 객체의 생성을 유연하게 해주는 패턴.

객체의 생성과정과 객체의 표현 방법을 분리.

2. Singleton : 전역 변수를 사용하지 않고 객체를 하나만 생성해 생성된 객체를 어디서든 참조할 수 있게 하는 패턴.

3. Factory : 객체 생성을 캡슐화하는 패턴.

팩토리 메소드 패턴 : 객체 생성을 위한 인터페이스를 정의하는데, 어떤 클래스의 인스턴스를 만들지는 서브 클래스에서 결정.

추상 팩토리 패턴 : 인터페이스를 이용해 서로 연관된, 또는 의존하는 객체를 구상 클래스를 지정하지 않고 생성.

**- Game Loop \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**- 수업 중 진행했던 내용들에 대한 Review**

**1. ImageSwitcher**

**- Button**

Android:onClick=”함수이름” 사용해 클릭 시 코드 진행되도록 함.

**- LinearLayout**

뷰를 수평 또는 수직 방향으로 배치할 수 있는 레이아웃.

Android:orientation=” “으로 vertical(수직), horizontal(수평)

**- interface as parameter**

**- ActionBar 제거**

ActionBar actionBar = getSupportActionBar();

actionBar.hide();

[res/values/styles.xml]

<resources>

<style name=”AppTheme” parent=”Theme.AppCompat.Light.DarkActionBar”> // Theme.AppCompat.Light.NoActionBar로 변경.

**- SharedPreferences(데이터 저장)**

public class MainActivity extends AppCompatActivity {

…

SharedPreferences pref;

SharedPreferences.Editor editor;

@Override

protected void onCreate(Bundle savedInstanceState) {

…

setPage(1);

pref = getSharedPreferences("pref", Activity.MODE\_PRIVATE);

// getSharedPreferences : 특정 이름을 가진 SharedPreferences 생성

// MODE\_PRIVATE : 해당 앱에서만 접근 가능

// MODE\_WORLD\_READABLE, MODE\_WORLD\_WRITEABLE : 다른 앱에서 접근 가능

editor = pref.edit();

page = pref.getInt("page", 1); // getInt(), getString()을 통해 저장된 데이터를 불러옴

setPage(page);

}

private void setPage(int newPage) {

…

editor.putInt("page", page); // putInt(), putString()을 통해 데이터 저장.

editor.apply();

}

}

**2. Cards**

**- Image Resource**

.setImageResource(resId); // resId : 카드 이미지 리소스(R.mipmap.card)

**- Toast**

Toast.makeText(this, R.string.same\_card, Toast.LENGTH\_SHORT).show(); …

// Toast.LENGTH\_SHORT : 짧게 Toast 메시지 표시 | Toast.LENGTH\_LONG : 길게 Toast 메시지 표시

**- AlertDialog**

private void askRetry() { // 재시작 알림창

new AlertDialog.Builder(this)

.setTitle("Restart?")

.setMessage("Do you really want to restart tha game?")

.setPositiveButton("Yes", new DialogInterface.OnClickListener() { // Yes 클릭할 경우

@Override

public void onClick(DialogInterface dialogInterface, int i) {

startGame(); // 재시작

}

})

.setNegativeButton("No", null) // No 클릭할 경우

.create()

.show();

}

private void askRetry() {

AlertDialog.Builder builder = new AlertDialog.Builder(this);

builder.setTitle("Restart?");

builder.setMessage("Do you really want to restart the game?");

builder.setPositiveButton("Yes", new DialogInterface.OnClickListener() {

@Override

public void onClick(DialogInterface dialogInterface, int i) {

startGame();

}

});

builder.setNegativeButton("No",null);

AlertDialog alert = builder.create();

alert.show();

} .show();

**- Multi-language**

[ strings.xml ]

<resources>

<string name="app\_name">카드게임</string>

<string name="Restart">재시작</string>

<string name="restart\_alert\_message">게임을 다시 시작하시겠습니까?</string>

<string name="yes">네</string>

<string name="no">아니오</string>

<string name="flips\_fmt">횟수: %d</string>

</resources>

코드에 있던 string을 R.string.xxx로 변경

**3. MoreControls**

**- Text Watcher**

1. 첫 번째 방법 : EditText에 TextWatcher를 추가

public class MainActivity extends AppCompatActivity {

…

@Override

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_main);

edit = findViewById(R.id.nameEdit);

output = findViewById(R.id.outputTextView);

edit.addTextChangedListener(new TextWatcher() { // 첫 번째 방법

@Override

public void beforeTextChanged(CharSequence charSequence, int i, int i1, int i2) {

Log.v(TAG, "before");

}

@Override

public void onTextChanged(CharSequence charSequence, int i, int i1, int i2) {

Log.d(TAG, "textChange: " + charSequence);

output.setText("Text Length: " + charSequence.length());

}

@Override

public void afterTextChanged(Editable editable) {

Log.v(TAG, "after");

}

});

}

2. 두 번째 방법 : TextWatcher를 MainActivity가 구현하는 방법

public class MainActivity extends AppCompatActivity implements TextWatcher {

private static final String TAG = MainActivity.class.getSimpleName();

private EditText edit;

private TextView output;

@Override

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_main);

edit = findViewById(R.id.nameEdit);

output = findViewById(R.id.outputTextView);

edit.addTextChangedListener(this);

}

// 두 번째 방법

@Override

public void beforeTextChanged(CharSequence charSequence, int i, int i1, int i2) {

Log.v(TAG, "before");

}

@Override

public void onTextChanged(CharSequence charSequence, int i, int i1, int i2) {

Log.d(TAG, "textChange: " + charSequence);

output.setText("Text Length: " + charSequence.length());

}

@Override

public void afterTextChanged(Editable editable) {

Log.v(TAG, "after");

}

}

3. 세 번째 방법 : TextWatcher를 member로 가지는 방법

public class MainActivity extends AppCompatActivity {

private static final String TAG = MainActivity.class.getSimpleName();

private EditText edit;

private TextView output;

@Override

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_main);

edit = findViewById(R.id.nameEdit);

output = findViewById(R.id.outputTextView);

edit.addTextChangedListener(editWatcher);

}

TextWatcher editWatcher = new TextWatcher() { // 세 번째 방법

@Override

public void beforeTextChanged(CharSequence charSequence, int i, int i1, int i2) {

Log.v(TAG, "before");

}

@Override

public void onTextChanged(CharSequence charSequence, int i, int i1, int i2) {

Log.d(TAG, "textChange: " + charSequence);

output.setText("Text Length: " + charSequence.length());

}

@Override

public void afterTextChanged(Editable editable) {

Log.v(TAG, "after");

}

};

}

**- Custom View**

public class GameView extends View {

private static final String TAG = GameView.class.getSimpleName();

public GameView(Context context) { // 생성자

super(context);

Log.d(TAG, "GameView cons");

}

public GameView(Context context, AttributeSet as) { // xml로 편집 가능하도록 하는 생성자

super(context, as);

Log.d(TAG, "GameView cons with as");

}

}

**- Paint**

public class GameView extends View {

private static final String TAG = GameView.class.getSimpleName();

private Paint paint;

public GameView(Context context) {

super(context);

Log.d(TAG, "GameView cons");

initView();

}

public GameView(Context context, AttributeSet as) {

super(context, as);

Log.d(TAG, "GameView cons with as");

initView();

}

private void initView() {

paint = new Paint();

paint.setColor(Color.parseColor("blue"));

}

**- RoundRect 그리기**

@Override

protected void onDraw(Canvas canvas) {

// super.onDraw(canvas);

int left = getPaddingLeft(); // 35

int top = getPaddingTop(); // 35

int right = getPaddingRight(); // 35

int bottom = getPaddingBottom(); // 35

int width = getWidth(); // 1300

int height = getHeight(); // 1691

float rx = width / 10, ry = height / 10;

canvas.drawRoundRect(left, top, width - right, height - bottom, rx, ry, paint); // Rect의 모서리를 rx, ry의 크기를 갖는 타원

**4. SampleGame**

**- Choreographer/VSync**

public class GameView extends View implements Choreographer.FrameCallback { // View 상속받음.

private static final String TAG = GameView.class.getSimpleName();

private Bitmap soccerBitmap; // 축구공 이미지.

private Rect srcRect = new Rect();

private Rect dstRect = new Rect();

private int ballDx, ballDy; // 축구공 이동 크기 나타내는 변수.

private long lastTimeNanos; // 기억하는 시각.

private int framePerSecond; // fps.

private Paint fpsPaint = new Paint(); // fps 그릴 때 사용할 Paint.

public GameView(Context context, @Nullable AttributeSet attrs) {

super(context, attrs);

initView(); // 초기화하는 함수.

Choreographer.getInstance().postFrameCallback(this);

}

@Override

public void doFrame(long currentTimeNanos) { // 매 프레임마다 불리는 함수

long now = currentTimeNanos; // 현재 시각.

int elapsed = (int) (now - lastTimeNanos); // 전 프레임부터 현재 프레임까지 흐른 시간 구함.

framePerSecond = 1\_000\_000\_000 / elapsed; // fps 구함.

lastTimeNanos = now; // 현재 시각 저장.

update(); // 게임 내용 업데이트하는 함수.

invalidate(); // 다시 그려지는 것을 예약하는 함수.

Choreographer.getInstance().postFrameCallback(this);

}

VSync : 주기적으로 신호를 발생시켜 화면을 바꿔주는 것

매 프레임마다 불리는 함수 사용

**- Touch event**

@Override

public boolean onTouchEvent(MotionEvent event) {

if(event.getAction() == MotionEvent.ACTION\_DOWN) { // 터치했을 경우 해당 위치로 이동하도록.

float x = event.getX();

float y = event.getY();

fighter.setPosition(x, y);

}

return super.onTouchEvent(event)

}

**- FPS**

public class GameView extends View { // View 상속받음.

…

private long lastTimeMillis; // 기억하는 시각.

private int framePerSecond; // fps.

private Paint fpsPaint = new Paint(); // fps 그릴 때 사용할 Paint.

public GameView(Context context, @Nullable AttributeSet attrs) {

super(context, attrs);

…

}

private void updateGame() { // View를 다시 그리는 함수.

long now = System.currentTimeMillis(); // 현재 시각.

int elapsed = (int) (now - lastTimeMillis); // 전 프레임부터 현재 프레임까지 흐른 시간 구함.

framePerSecond = 1000/elapsed; // fps 구함.

lastTimeMillis = now; // 현재 시각 저장.

…

**- Handler**

public GameView(Context context, @Nullable AttributeSet attrs) {

super(context, attrs);

initView(); // 초기화하는 함수.

handler = new Handler();

updateGame(); // View를 다시 그리는 함수.

}

private void updateGame() { // View를 다시 그리는 함수.

this.invalidate(); // 다시 그려지는 것을 예약하는 함수.

handler.post(new Runnable() { // 할 일을 한 후 updateGame() 호출되도록.

@Override

public void run() {

updateGame(); // handler 리턴 후 시간이 지난 후 updateGame() 호출됨.

}

});

}

**- Singleton** : 어떤 클래스가 최초 한번만 메모리를 할당하고 그 메모리에 인스턴스를 만들어 사용하는 디자인 패턴

public class MainGame {

private static MainGame singleton;

public static MainGame getInstance() {

if(singleton == null) {

singleton = new MainGame();

}

return singleton;

}

}

**- Rotation**

public class Fighter implements GameObject {

…

private float angle; // 비행기 각도.

public Fighter(float x, float y) {

this.x = x; // 초기값 설정.

this.y = y;

radius = Metrics.size(R.dimen.fighter\_radius);

dstRect.set(x - radius, y - radius, x + radius, y + radius);

this.tx = x;

this.ty = y;

targetRect.set(dstRect);

angle = -(float) (Math.PI / 2);

…

}

public void draw(Canvas canvas) { // 그리는 함수.

canvas.save();

canvas.rotate((float) (angle \* 180 / Math.PI) + 90, x, y);

canvas.drawBitmap(bitmap, null, dstRect, null);

canvas.restore();

if (dx != 0 && dy != 0) {

canvas.drawBitmap(targetBitmap, null, targetRect, null);

}

}

public void update() {

…

}

public void setTargetPosition(float tx, float ty) {

this.tx = tx;

this.ty = ty;

targetRect.set(tx - radius/2, ty - radius/2, tx + radius/2, ty + radius/2);

angle = (float) Math.atan2(ty - y, tx - x);

float speed = Metrics.size(R.dimen.fighter\_speed);

float dist = speed \* MainGame.getInstance().frameTime;

dx = (float) (dist \* Math.cos(angle));

dy = (float) (dist \* Math.sin(angle));

}

}

**- Metrics**

public class Fighter implements GameObject {

…

public Fighter(int x, int y) {

…

// float radius = res.getDimension(R.dimen.fighter\_radius);

float radius = Metrics.size(R.dimen.fighter\_radius);

dstRect.set(x - radius, y - radius, x + radius, y + radius);

if (bitmap == null) { // 리소스 한번만 로드하도록.

Resources res = GameView.view.getResources();

bitmap = BitmapFactory.decodeResource(res, R.mipmap.plane\_240);

srcRect.set(0, 0, bitmap.getWidth(), bitmap.getWidth()); // srcRect 초기화.

}

}

public void draw(Canvas canvas) { // 그리는 함수.

canvas.drawBitmap(bitmap, srcRect, dstRect, null);

}

public void update() {

float angle = (float) Math.atan2(ty - y, tx - x); // 현재 위치와 타겟 위치의 각도 구함.

// float speed = 1000;

float speed = Metrics.size(R.dimen.fighter\_speed);

…

[ Metrics.java ]

public class Metrics {

public static float size(int dimenRedId) {

Resources res = GameView.view.getResources();

float size = res.getDimension(dimenRedId);

return size;

}

}

**5. DragonFlight**

**- BitmapPool**

public class BitmapPool {

private static HashMap<Integer, Bitmap> bitmaps = new HashMap<>();

public static Bitmap get(int mipmapResId) {

Bitmap bitmap = bitmaps.get(mipmapResId);

if (bitmap == null) {

Resources res = GameView.view.getResources();

bitmap = BitmapFactory.decodeResource(res, mipmapResId);

bitmaps.put(mipmapResId, bitmap);

}

return bitmap;

}

}

**- package refactor**

**- dimens.xml**

[ dimens.xml ]

<resources>

<dimen name="fighter\_radius">30dp</dimen>

<dimen name="fighter\_speed">300dp</dimen>

<dimen name="ball\_radius">20dp</dimen>

<dimen name="ball\_speed\_min">20dp</dimen>

<dimen name="ball\_speed\_max">200dp</dimen>

<dimen name="laser\_length">20dp</dimen>

<dimen name="laser\_speed">1000dp</dimen>

<dimen name="laser\_width">3dp</dimen>

</resources>

R.dimen.xxx로 사용

**- Activity Life-cycle**

**- Collision**

[ CollisionHelper.java ]

public class CollisionHelper {

public static boolean collides(BoxCollidable o1, BoxCollidable o2) {

RectF r1 = o1.getBoundingRect();

RectF r2 = o2.getBoundingRect();

if(r1.left > r2.right) { return false; }

if(r1.top > r2.bottom) { return false; }

if(r1.right < r2.left) { return false; }

if(r1.bottom < r2.top) { return false; }

return true;

}

}

[ BocCollidable.java ]

public interface BoxCollidable {

public RectF getBoundingRect();

}

[ MainGame.java ]

public class MainGame {

…

public void init() {

…

collisionPaint = new Paint();

collisionPaint.setStyle(Paint.Style.STROKE);

collisionPaint.setColor(Color.RED);

}

…

public void draw(Canvas canvas) {

for (GameObject gobj : gameObjects) {

gobj.draw(canvas);

if(gobj instanceof BoxCollidable) { // 바운딩 박스 그리기.

RectF box = ((BoxCollidable) gobj).getBoundingRect();

canvas.drawRect(box, collisionPaint);

}

}

}

public void update(int elapsedNanos) {

…

checkCollision();

}

private void checkCollision() {

for(GameObject o1 : gameObjects) {

if(!(o1 instanceof Enemy)) { // Enemy가 아닌 경우 무시.

continue;

}

Enemy enemy = (Enemy) o1;

boolean removed = false;

for(GameObject o2 : gameObjects) {

if(!(o2 instanceof Bullet)) { // Bullet이 아닌 경우 무시.

continue;

}

Bullet bullet = (Bullet) o2;

if(CollisionHelper.collides(enemy, bullet)) { // enemy와 bullet이 충돌했을 경우

Log.d(TAG, "Collision");

remove(bullet);

remove(enemy);

removed = true;

break;

}

}

if(removed) {

continue;

}

}

}