Chapter 05

게임 프레임워크

(Game Framework)

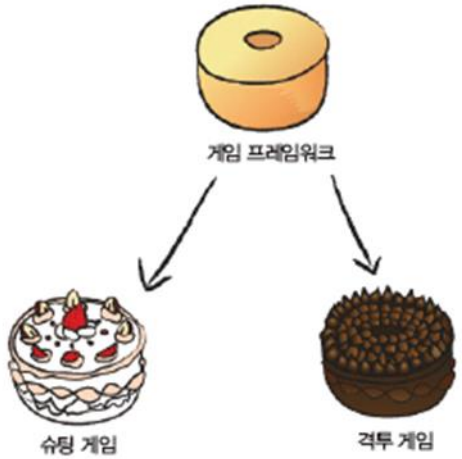
Contents

5. 1 프레임워크 개요

❖ 게임 프레임 워크란?

◆ 설명

- ✓ 게임 프레임워크는 게임 제작에서 가장 큰 뼈대 역할
- ✓ 게임 프레임워크를 만든 뒤에는 제작할 게임에 따른 요소와 로직만 추가해주면 게임을 제작할 수 있음
- ✓ 아래 그림에서 보는 것처럼 프레임워크는 베이스 역할



5.2 게임 제작

❖ 게임 제작

◆ GameView 클래스 & GameActivity 클래스

```
public class GameView extends View  {
    public GameView (Context context)  {
            super (context);
    }
    @Override
    public void onDraw (Canvas canvas)  {
            // 작동 여부 확인용 그림
            Bitmap _scratch = BitmapFactory.decodeResource(getResource( ),
                                R.drawable. icon);
            canvas.drawColor(Color. BLACK);
            canvas.drawBitmap(_scratch, 10, 10, null);
    }
}
```

5.2 게임 제작

❖ 게임 제작 (cont.)

◆ GameView 클래스 & GameActivity 클래스

```
public class GameActivity extends Acitivity  {
    @Override
    public void onCreate (Bundle savedInstanceState)  {
            super .onCreate (savedInstanceState);
            setContentView (new GameView( this ));
    }
}
```

5. 3 SurfaceView

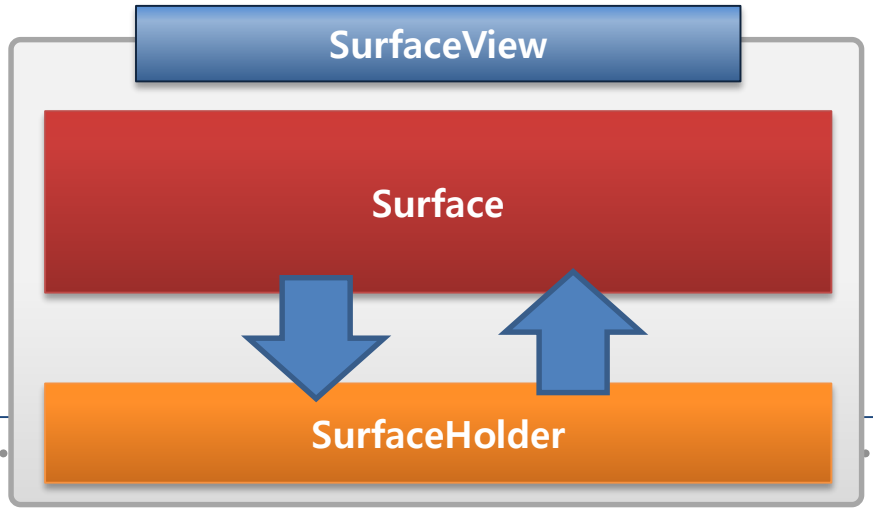
❖ SurfaceView를 이용한 빠른 그래픽 처리 기반

◆ 안드로이드에서

- ✓ 5초 이상 동작이 멈추었을 때 ANR(Application Not Responding) 에러 발생
- ✓ 게임의 경우 모든 연산을 UI 스레드에서 처리하다 보면 ANR이 발생

◆ SurfaceView

- ✓ View는 화면 업데이트를 UI 스레드에서 처리
- ✓ SurfaceView는 화면 업데이트를 백그라운드로 처리
- ✓ 구조



5. 3 SurfaceView

❖ SurfaceView를 이용한 빠른 그래픽 처리 기반 (cont.)

◆ SurfaceView의 적용

✓ GameView 클래스가 상속받는 클래스를 SurfaceView로 변경

```
public class GameView extends SurfaceView {
    public GameView (Context context) {
            super (context);
    }
    @Override
    public void onDraw (Canvas canvas) {
    }
}
```

→ SurfaceHolder.Callback의 순수 가상 메서드 구현 필요

5. 3 SurfaceView

❖ SurfaceView를 이용한 빠른 그래픽 처리 기반 (cont.)

◆ SurfaceView의 적용

```
public class GameView extends SurfaceView implements SurfaceHolder.Callback {
    public GameView (Context context)  {
            super (context);
            getHolder( ).addCallback( this );
    }
    @Override
    public void surfaceChanged (SurfaceHolder holder, int format, int width, int height)
    {  }
    @Override
    public void surfaceCreated (SurfaceHolder holder)
    {  }
    @Override
    public void surfaceDestroyed (SurfaceHolder holder)
    {  }
    … …
}
```

5. 3 SurfaceView

❖ SurfaceView를 이용한 빠른 그래픽 처리 기반 (cont.)

◆ 그림 관리 클래스

```
public class GameViewThread extends Thread  {
    // 접근을 위한 멤버 변수
    private SurfaceHolder m_surfaceHolder;
    private GameView m_gameView;
    // 스레드 실행 상태 멤버 변수
    private boolean m_run = false;
    public GameViewThread(SurfaceHolder surfaceHolder, GameView gameView)  {
            m_surfaceHolder = surfaceHolder;
            m_gameView = gameView;
    }
    public void setRunning( boolean run )  {
            m_run = run;
    }
    @Override
    public void run( )  {    }
}
```

5. 3 SurfaceView

❖ SurfaceView를 이용한 빠른 그래픽 처리 기반 (cont.)

◆ 그림 관리 클래스

✓ GameViewThread 클래스를 GameView 클래스의 멤버로 추가하고, 생성자에서 인스턴스를 생성하는 코드 작성

```
public class GameView extends SurfaceView implements SurfaceHolder.Callback  {
    private GameViewThread m_thread;

    public GameView(Context context)  {
            super (context);
            getHolder( ).addCallback( this );
            m_thread = new GameViewThread(getHolder( ), this);
    }
    … …
}
```

5. 3 SurfaceView

❖ SurfaceView를 이용한 빠른 그래픽 처리 기반 (cont.)

◆ 그림 관리 클래스

✓ SurfaceHolder.Callback을 implements해서 추가하게 된 sufraceCreated 메서드와 surfaceDestroyed 메서드에 스레드를 시작하고 종료하는 루틴 작성

```
public class GameView extends SurfaceView implements SurfaceHolder.Callback  {
    … …
    @Override
    public void surfaceCreated(SurfaceHolder arg0)  {
            // 스레드를 실행 상태로 만든다.
            m_thread .setRunning( true );
            // 스레드 실행
            m_thread .start( );
    }
```

5. 3 SurfaceView

```
    @Override
    public void surfaceDestroyed(SurfaceHolder arg0)  {
            boolean retry = true;
            m_thread .setRunning( false );
            while (retry)  {
                try  {
                        // 스레드를 중지시킨다.
                        m_thread .join( );
                        retry = false;
                } catch (InterruptedException e)  {
                        // 스레드가 종료되도록 계속 시도
                }
            }
    }
}
```

5. 3 SurfaceView

❖ SurfaceView를 이용한 빠른 그래픽 처리 기반 (cont.)

◆ 지금까지

- ✓ GameViewThread를 멤버 변수로 추가하고, 생성자에서 인스턴스 초기화하고, GameView의 Surface가 생성될 때 스레드를 실행하고, Surface가 파괴될 때 스레드를 종료시키는 루틴 구현

◆ 이제

- ✓ GameViewThread 클래스의 run 메서드를 구현
- ✓ run 메서드에서는 캔버스 객체에 접근

- ✓ 여기까지 작성하고 컴파일하고 실행

5. 3 SurfaceView

```
public class GameViewThread extends Thread  {
    … …
    @Override
    public void run( )  {
            Canvas _canvas;
            while ( m_run )  {
                _canvas = null;
                try  {    // SurfaceHolder를 통해 Surface에 접근해서 가져옴
                        _canvas = m_surfaceHolder .lockCanvas( null );
                        synchronized ( m_surfaceHolder )  {
                            m_gameView .onDraw(_canvas);  // 그림을 그림
                        }
                } finally  {
                        if (_canvas != null )
                            // Surface를 화면에 표시함
                            m_surfaceHolder .unlockCanvasAndPost(_canvas);
                }
            }
    }
}
```

5. 3 SurfaceView

❖ SurfaceView를 이용한 빠른 그래픽 처리 기반 (cont.)

◆ 데이터의 자동 갱신

```
public class GameView extends SurfaceView implements SurfaceHolder.Callback {
    … …
    public void Update( ) {

    }
    … …
}
```

✓ Update 메서드를 스레드에서 지속적으로 실행해야만 갱신이 수행되므로 GameViewThread의 run 메서드에서 Update 메서드를 실행하게 함

◆ 여기까지

✓ 화면과 관련된 이벤트가 발생하지않아도 게임 루프가 계속 돌아가는 기반 작성

5. 3 SurfaceView

```
public class GameViewThread extends Thread  {
    … …
    @Override
    public void run( )  {
            Canvas _canvas;
            while ( m_run )  {
                _canvas = null;
                try  {
                        m_gameView .Update( );
                        _canvas = m_surfaceHolder .lockCanvas( null );
                        synchronized ( m_surfaceHolder )  {
                            m_gameView .onDraw(_canvas);
                        }
                } finally  {
                        if (_canvas != null )
                            m_surfaceHolder .unlockCanvasAndPost(_canvas);
                }
            }
    }
}
```

5.4 게임 프레임워크

❖ 애플리케이션의 모든 것을 관리하는 AppManager

◆ AppManager 클래스 생성

- ✓ New – Class → AppManager 클래스 생성
- ✓ AppManager 클래스는 이 프레임워크를 사용하는 애플리케이션을 관리
- ✓ SingleTon 패턴 적용하여 프로젝트 어디서나 접근 가능하도록 작성

```
public class AppManager  {
    private static AppManager s_instance;

    public static AppManager getInstance( )  {
            if (s_instance == null)  s_instance = new AppManager( );
            return s_instance;
    }
}
```

5.4 게임 프레임워크

❖ 애플리케이션의 모든 것을 관리하는 AppManager (cont.)

◆ AppManager 클래스

- ✓ 게임에서 사용할 서브시스템이나 애플리케이션 관련 정보를 멤버 변수로 추가
- ✓ 게임 뷰나 리소스를 추가

5. 4 게임 프레임워크

```
public class AppManager  {
    private GameView m_gameView;
    private Resources m_resources;

    void setGameView (GameView _gameView)  {
            m_gameView = _gameView;
    }
    void setResources (Resources _resources)  {
            m_resources = _resources;
    }
    void GameView getGameView( )  {
            return m_gameView;
    }
    void Resources getResource( )  {
            return m_resources;
    }
    … …
}
```

5.4 게임 프레임워크

❖ 애플리케이션의 모든 것을 관리하는 AppManager (cont.)

◆ AppManager 클래스

✓ AppManager에서 GameView와 Resources의 인스턴스 정보를 알려면 GameView 생성자에서 AppManager에 인스턴스 값을 넘겨줘야 함

```
public GameView(Context context) {
    super (context);
    // 키 입력 처리를 받기 위해서
    setFocusable( true );

    AppManager.getInstance( ).setGameView( this );
    AppManager.getInstance( ).setResources(getResources( ));

    getHolder.addCallback( this );
}
```

5.4 게임 프레임워크

❖ 애플리케이션의 모든 것을 관리하는 AppManager (cont.)

◆ AppManager 클래스

- ✓ 여기에 비트맵 관련 기능도 추가
- ✓ 매니저 클래스를 이용해 비트맵을 가져오는 getBitmap 메서드를 작성

```
public class AppManager  {
    public Bitmap getBitmap( int r )  {
            return BitmapFactory.decodeResource( m_resources, r);
    }
    … …
}
```

5.4 게임 프레임워크

❖ 애플리케이션의 모든 것을 관리하는 AppManager (cont.)

◆ AppManager 클래스

- ✓ 여기서는 GameView와 Resource 접근을 위해서만 사용
- ✓ 상용 수준의 게임에서는 애플리케이션 실행 정보나 여러 가지 정보를 관리하는 기능을 추가
- ✓ 전체 코드

5. 4 게임 프레임워크

```
public class AppManager  {
    // Main GameView
    private GameView m_gameView;
    // Main GameView의 Resources
    private Resources m_resources;

    void setGameView (GameView _gameView)  {
            m_gameView = _gameView;
    }
    void setResources (Resources _resources)  {
            m_resources = _resources;
    }
    void GameView getGameView( )  {
            return m_gameView;
    }
    void Resources getResource( )  {
            return m_resources;
    }
```

5. 4 게임 프레임워크

```
    public Bitmap getBitmap( int r )  {
            return BitmapFactory.decodeResource( m_resources, r);
    }

    private static AppManager s_instance;

    public static AppManager getInstance( )  {
            if (s_instance == null)  s_instance = new AppManager( );
            return s_instance;
    }
}
```

5.4 게임 프레임워크

❖ SoundManager를 통한 사운드 기반 구현

◆ SoundManager 클래스

- ✓ New – Class → SoundManager 클래스 생성
- ✓ 여러 사운드 동시 출력에 문제없고, 게임 제작에 적합한 SoundPool을 이용
- ✓ SingleTon 패턴 적용하여 프로젝트 어디서나 접근 가능하도록 작성

```
public class SoundManager  {
    private static SoundManager s_instance;

    public static SoundManager getInstance( )  {
            if (s_instance == null)  s_instance = new SoundManager( );
            return s_instance;
    }
}
```

5.4 게임 프레임워크

❖ SoundManager를 통한 사운드 기반 구현 (cont.)

◆ 기본적인 사운드 처리 구현

```
public class SoundManager {
    // 멤버 변수
    private SoundPool       m_SoundPool;
    private HashMap         m_SoundPoolMap;
    private AudioManager    m_AudioManager;
    private Context         m_Activity;
    … …
}
```

- m_SoundPool : 안드로이드에서 지원하는 사운드풀
- m_SoundPoolMap : 불러온 사운드의 아이디 값을 지정할 해시맵
- m_AudioManager : 사운드 관리를 위한 오디오 매니저
- m_Activity : 애플리케이션의 컨텍스트 값

5.4 게임 프레임워크

❖ SoundManager를 통한 사운드 기반 구현 (cont.)

◆ 기본적인 사운드 처리 구현

✓ 멤버 변수 초기화

```
public class SoundManager  {
    … …
    public void Init (Context _context)  {
            // 멤버 변수 생성과 초기화
            m_SoundPool = new SoundPool(4, AudioManager. STREAM_MUSIC, 0);
            m_SoundPoolMap = new HashMap( );
            m_AudioManager = (AudioManager)_context.getSystemService
                                    (Context. AUDIO_SERVICE);
            m_Activity = _context;
    }
    … …
}
```

5.4 게임 프레임워크

❖ SoundManager를 통한 사운드 기반 구현 (cont.)

◆ 기본적인 사운드 처리 구현

✓ 사운드를 불러오는 addSound 메서드

```
public class SoundManager  {
    … …
    public void addSound( int _index, int _soundID)  {
            int id = m_SoundPool .load( m_Activity, _soundID, 1);  // 사운드를 로드
            m_SoundPoolMap .put(_index, id); // 해시맵에 아이디 값을 받아온 인덱스저장
    }
    … …
}
```

5.4 게임 프레임워크

❖ SoundManager를 통한 사운드 기반 구현 (cont.)

◆ 기본적인 사운드 처리 구현

✓ 게임에서 사용하는 사운드 재생

- 효과음 : 재생시간이 짧고, 재생이 끝나지 않아도 다시 재생 가능 (반복X)
- 배경음 : 재생시간이 길고, 소리 하나만 재생 (반복O)

✓ 사운드의 인덱스값을 받는 두가지 메서드

- play
- playLooped

5. 4 게임 프레임워크

```
public class SoundManager  {
    … …
    public void play( int _index)  {
            // 사운드 재생
            float streamVolume = m_AudioManager .getStreamVolume
                                    (AudioManager. STREAM_MUSIC);
            streamVolume = streamVolume / m_AudioManager .getStreamMaxVolume
                                    (AudioManager. STREAM_MUSIC);
            m_SoundPool .play((Integer) m_SoundPoolMap .get(_index),
                                    streamVolume, streamVolume, 1, 0, 1f);
    }
    … …
}
```

5. 4 게임 프레임워크

```
public class SoundManager  {
    … …
    public void playLooped( int _index)  {
            // 사운드 반복 재생
            float streamVolume = m_AudioManager .getStreamVolume
                                    (AudioManager. STREAM_MUSIC);
            streamVolume = streamVolume / m_AudioManager .getStreamMaxVolume
                                    (AudioManager. STREAM_MUSIC);
            m_SoundPool .play((Integer) m_SoundPoolMap .get(_index),
                                    streamVolume, streamVolume, 1, -1, 1f);
    }
    … …
}
```

5.4 게임 프레임워크

❖ SoundManager를 통한 사운드 기반 구현 (cont.)

◆ 사운드 매니저 클래스의 사용

✓ 사운드 매니저 사용을 위한 초기화

```
SoundManager.getInstance( ).Init(액티비티값);
```

✓ 사운드 리소스를 불러오는 과정

```
SoundManager.getInstance( ).addSound(1, R.raw.sound1);
SoundManager.getInstance( ).addSound(1, R.raw.sound2);
SoundManager.getInstance( ).addSound(1, R.raw.sound3);
```

```
SoundManager.getInstance( ).play(1);
SoundManager.getInstance( ).playLooped(2);
SoundManager.getInstance( ).play(3);
```

5.4 게임 프레임워크

❖ SoundManager를 통한 사운드 기반 구현 (cont.)

◆ 기본적인 사운드 처리 구현

✓ 인덱스 상수화

```
public static final int SOUND_EFFECT_1 = 1;
public static final int SOUND_EFFECT_2 = 2;
public static final int SOUND_EFFECT_3 = 3;

// 사운드 리소스 불러오기
SoundManager.getInstance( ).addSound( SOUND_EFFECT_1, R.raw.sound1);
SoundManager.getInstance( ).addSound( SOUND_EFFECT_2, R.raw.sound1);
SoundManager.getInstance( ).addSound( SOUND_EFFECT_3, R.raw.sound1);
// 사운드 재생
SoundManager.getInstance( ).play(SOUND_EFFECT_1);
SoundManager.getInstance( ).playLooped(SOUND_EFFECT_2);
SoundManager.getInstance( ).play(SOUND_EFFECT_3);
```

5.4 게임 프레임워크

❖ SoundManager를 통한 사운드 기반 구현 (cont.)

◆ SoundManager 전체 코드

```
public class SoundManager  {
    // 멤버 변수
    private SoundPool           m_SoundPool;
    private HashMap             m_SoundPoolMap;
    private AudioManager        m_AudioManager;
    private Context             m_Activity;

    public void Init (Context _context)  {
            // 멤버 변수 생성과 초기화
            m_SoundPool = new SoundPool(4, AudioManager. STREAM_MUSIC, 0);
            m_SoundPoolMap = new HashMap( );
            m_AudioManager = (AudioManager)_context.getSystemService
                                (Context. AUDIO_SERVICE);
            m_Activity = _context;
    }
```

5. 4 게임 프레임워크

```
public void addSound( int _index, int _soundID)  {
        int id = m_SoundPool .load( m_Activity, _soundID, 1);  // 사운드를 로드
        m_SoundPoolMap .put(_index, id); // 해시맵에 아이디 값을 받아온 인덱스저장
}
public void play( int _index)  {
        // 사운드 재생
        float streamVolume = m_AudioManager .getStreamVolume
                                (AudioManager. STREAM_MUSIC);
        streamVolume = streamVolume / m_AudioManager .getStreamMaxVolume
                                (AudioManager. STREAM_MUSIC);
        m_SoundPool .play((Integer) m_SoundPoolMap .get(_index),
                                streamVolume, streamVolume, 1, 0, 1f);
}
```

5. 4 게임 프레임워크

```
    public void playLooped( int _index)  {
            // 사운드 반복 재생
            float streamVolume = m_AudioManager .getStreamVolume
                                    (AudioManager. STREAM_MUSIC);
            streamVolume = streamVolume / m_AudioManager .getStreamMaxVolume
                                    (AudioManager. STREAM_MUSIC);
            m_SoundPool .play((Integer) m_SoundPoolMap .get(_index),
                                    streamVolume, streamVolume, 1, -1, 1f);
    }

    private static SoundManager s_instance;

    public static SoundManager getInstance( )  {
            if (s_instance == null)  s_instance = new SoundManager( );
            return s_instance;
    }
}
```

5.4 게임 프레임워크

❖ 게임 상태의 이해와 게임 상태 기반 구현

◆ 게임 상태에 대한 이해

- ✓ 예) 짝맞추기 게임
  - STATE_READY 게임 시작 전 준비 상태
  - STATE_GAME 게임 중
  - STATE_END 게임 종료
- ✓ 상태에 대한 효율적 관리 위해 → State 패턴 적용

5.4 게임 프레임워크

❖ 게임 상태의 이해와 게임 상태 기반 구현 (cont.)

◆ 슈퍼 클래스 ; IState 클래스

✓ 일반적 게임에서의 인터페이스 클래스의 틀

```
public interface IState  {
    public void Init( );
            // 이 상태로 바뀌었을 때 실행할 것들
    public void Destroy( );
            // 다른 상태로 바뀌었을 때 실행할 것들
    public void Update( );
            // 지속적으로 수행할 것들
    public void Render(Canvas canvas);
            // 그려야 할 것들
}
```

5.4 게임 프레임워크

❖ 게임 상태의 이해와 게임 상태 기반 구현 (cont.)

◆ 슈퍼 클래스 ; IState 클래스

✓ 안드로이드 기반에서의 틀

```
public interface IState  {
    public void Init( );
            // 상태가 생성되었을 때
    public void Destroy( );
            // 상태가 소멸될 때
    public void Update( );
            // 지속적으로 수행할 것들
    public void Render(Canvas canvas);
            // 그려야 할 것들
    public boolean onKeyDown( int keyCode, KeyEvent evnet);
            // 키 입력 처리
    public boolean onTouchEvent(MotionEvent event);
            // 터치 입력 처리
}
```

5.4 게임 프레임워크

❖ 게임 상태의 이해와 게임 상태 기반 구현 (cont.)

◆ GameView에 상태 추가

```
public class GameView extends SurfaceView  implements SurfaceHolder.Callback  {
    private IState m_state;
    @Override
    public void onDraw (Canvas canvas)  {
            canvas.draw(Color. BLACK);
            m_state .Render(canvas);        }
    public void Update( )  {        m_state .Update( );     }
    @Override
    public void onKeyDown (int keyCode, KeyEvent event)  {
            m_state .onKeyDown(keyCode, event);                }
    @Override
    public void onTouchEvent (MotionEvent event)  {
            m_state .onTouchEvent(event);                }
    … …
}
```

5.4 게임 프레임워크

❖ 게임 상태의 이해와 게임 상태 기반 구현 (cont.)

◆ GameView에 상태 추가

✓ 상태 변경 메서드 추가

```
public class GameView extends SurfaceView  implements SurfaceHolder.Callback  {
    public void changeGameState(IState _state)  {
            if ( m_state != null)
                m_state .Destroy( );
            _state.Init( );
            m_state = _state;
    }
    … …
}
```

5.4 게임 프레임워크

❖ 게임 상태의 이해와 게임 상태 기반 구현 (cont.)

◆ 게임 상태 기반

✓ IntroState 클래스

- IState 클래스를 상속
- 화면을 조금 꾸며주는 메서드 추가

```
public class IntroState implements IState  {
    Bitmap icon;
    int x, y;

    @Override
    public void Destroy( )  {        }
    @Override
    public void Init( )  {
            icon = AppManager.getInstance( ).getBitmap(R.drawable. icon);
    }
```

5. 4 게임 프레임워크

```
    @Override
    public void Render(Canvas canvas)  {
            canvas.drawBitmap( icon, x, y, null);
    }
    @Override
    public void Update( )  {        }
    @Override
    public boolean onKeyDown( int keyCode, KeyEvent event)  {
            return true;
    }
    @Override
    public boolean onTouchEvent(MotionEvent event)  {
            return true;
    }
}
```

5.4 게임 프레임워크

❖ 게임 상태의 이해와 게임 상태 기반 구현 (cont.)

◆ GameView 클래스

- ✓ 첫 게임 상태 값으로 IntroState의 인스턴스를 넘기도록 수정
- ✓ 실행하여 아이콘 표시 확인

```
public GameView(Context context)  {
    super (context);
    // 키 입력 처리를 받기 위해서
    setFocusable( true );
    AppManager.getInstance( ).setGameView( this );
    AppManager.getInstance( ).setResources(getResources( ));
    getHolder.addCallback( this );

    ChangeGameState( new IntroState( ));
}
```

5.4 게임 프레임워크

❖ 게임 상태의 이해와 게임 상태 기반 구현 (cont.)

◆ CreditSate 상태 추가

- ✓ 이미지(android.png) 리소스 폴더에 추가
- ✓ IState 클래스를 상속

```
public class CreditState implements IState  {
    Bitmap android;
    int x, y;

    @Override
    public void Destroy( )  {        }
    @Override
    public void Init( )  {
            icon = AppManager.getInstance( ).getBitmap(R.drawable. android);
    }
```

5. 4 게임 프레임워크

```
    @Override
    public void Render(Canvas canvas)  {
            canvas.drawBitmap( android, x, y, null);
    }
    @Override
    public void Update( )  {        }
    @Override
    public boolean onKeyDown( int keyCode, KeyEvent event)  {
            return true;
    }
    @Override
    public boolean onTouchEvent(MotionEvent event)  {
            return true;
    }
}
```

5.4 게임 프레임워크

❖ 게임 상태의 이해와 게임 상태 기반 구현 (cont.)

◆ CreditSate 상태 추가

✓ 동작 확인 위해 GameView에서 상태 변경

```
public GameView(Context context) {
    … …

    ChangeGameState( new CreditState( ));
}
```

✓ 동작 확인 (→ 이후 다시 IntroState로 변경)

5.4 게임 프레임워크

❖ 게임 상태의 이해와 게임 상태 기반 구현 (cont.)

◆ 상태 기반 구현

- ✓ 상태 변경 : IntroState → 화면 터치 → CreditState
- ✓ 터치 이벤트가 발생하면 세임의 상태를 CreditState로 변경

```
public class IntroState implements IState  {
    … …
    @Override
    public boolean onTouchEvent(MotionEvent event)  {
            AppManager.getInstance( ).getGameView( ).ChangeGameState
                    ( new CreditState( ));
            return true;
    }
}
```

5.4 게임 프레임워크

❖ 게임 상태의 이해와 게임 상태 기반 구현 (cont.)

◆ 상태 기반 구현

- ✓ 상태 기반 구현시 ChangeState 메서드로 게임 상태를 변경
  - 무조건적인 상태 변경
- ✓ 대부분 게임은 상태 변경 후 다시 돌아오면 이전 상태를 그대로 유지

◆ 스택을 이용한 게임 상태 관리

- ✓ IState 인터페이스에 Pause, Resume 메서드를 추가
  - 다른 상태로 변경되거나 다시 돌아올 때 호출
- ✓ ChangeState 외에 PushState, PopSate 메서드를 추가
  - PushState 게임 상태의 이동, 현재 상태를 메모리에 유지한 채로 다음 상태로 이동
  - PopSate 이전 상태로 이동, 현재 상태는 소멸

5. 4 게임 프레임워크

❖ 게임에 사용할 프레임워크의 부가 요소 제작

◆ 게임의 경우

- ✓ 그림 하나에 대한 좌표나 비트맵분 아니라 여러가지 멤버 변수 선언
- ✓ 게임은 한 화면에 그림을 수십 개씩 그려야 함
- ✓ 그림을 표시하는 데 필요한 멤버 변수와 처리 과정을 하나의 클래스로 만들면 편리

5.4 게임 프레임워크

❖ 게임에 사용할 프레임워크의 부가 요소 제작 (cont.)

◆ 게임 화면의 구성

✓ 게임 화면에 표시되는 대부분을 차지하는 GraphicObject

- 그림을 나타내는 클래스 구성
- New – Class → GraphicObject 클래스 생성

```
public class GraphicObject  {

}
```

```
public class GraphicObject  {
    private Bitmap      m_bitmap;
    private int         m_x;
    private int         m_y;
}
```

5.4 게임 프레임워크

❖ 게임에 사용할 프레임워크의 부가 요소 제작 (cont.)

◆ 게임 화면의 구성

✓ GraphicObject 클래스 생성자 추가

```
public class GraphicObject  {
    … …
    public GraphicObject(Bitmap bitmap)  {
            m_bitmap = bitmap;
            m_x = 0;
            m_y = 0;
    }          }
```

```
public class GraphicObject  {
    … …
    public void Draw(Canvas canvas)  {
            canvas.drawBitmap( m_bitmap, m_x, m_y, null);
    }          }
```

5.4 게임 프레임워크

❖ 게임에 사용할 프레임워크의 부가 요소 제작 (cont.)

◆ 게임 화면의 구성

✓ GraphicObject 클래스 사용

```
// GraphicObject의 생성
GraphicObject obj;
obj = new GraphicObject(비트맵데이터);

// GraphicObject의 사용
obj.Draw(canvas);
```

✓ GraphicObject 클래스의 구동 여부 확인

5. 4 게임 프레임워크

```
public class GameView extends SurfaceView implements SufaceHolder.Callback  {
    … …
    private GraphicObject m_Image;
    … …
    public GameView(Context context)  {
            super (context);
            setFocusable( true );
            AppManager.getInstance( ).setGameView( this );
            AppManager.getInstance( ).setResouces( getResources( ) );
            getHolder( ).addCallback( this );
            m_thread = new GameViewThread(getHolder( ), this);
            m_Image = new GraphicObject(AppManager.getInstance( )
                                        .getBitmap(R.drawable. background2));
    }
    @Override
    public void onDraw(Canvas canvas)  {
            canvas.draw(Color. BLACK);
            m_Image .Draw(canvas);
    }
    … …
}
```

5. 4 게임 프레임워크

❖ 게임에 사용할 프레임워크의 부가 요소 제작 (cont.)

◆ 게임 화면의 구성

✓ 그림의 위치 변경을 위한 메서드 추가

```
public class GraphiObject  {
    … …
    public void setPosition( int x, int y)  {
            m_x = x;
            m_y = y;
    }
}
```

✓ 전체 코드

```
public class GraphicObject  {
    private Bitmap      m_bitmap;              // 비트맵
    private int         m_x;                   // x, y 좌표
    private int         m_y;
    // 생성자
    public GraphicObject(Bitmap bitmap)  {
            m_bitmap = bitmap;
            m_x = 0;
            m_y = 0;
    }
    // 좌표 설정
    public void setPosition( int x, int y)  {
            m_x = x;
            m_y = y;
    }
    // 이미지 그림
    public void Draw(Canvas canvas)  {
            canvas.drawBitmap( m_bitmap, m_x, m_y, null);
    }
    // X, Y 각 좌표 반환
    public int getX( )  {    return m_x;    }
    public int getY( )  {    return m_y;    }
}
```

5.4 게임 프레임워크

❖ 게임에 사용할 프레임워크의 부가 요소 제작 (cont.)

◆ 애니메이션 구성

✓ 게임에서 움직임을 표현하는 SpriteAnimation

- 2D 스프라이트 애니메이션 구현
- 비트맵 하나를 리소스에 추가하고, 이 비트멥의 애니메이션 정보를 코드에 직접 작성하는 방식으로 애니메이션 구현
- SpriteAnimation 클래스 생성

```
public class SpriteAnimation extends GraphicObject {
    public SpriteAnimation(Bitmap bitmap) {
            super (bitmap);
    }
}
```

5.4 게임 프레임워크

❖ 게임에 사용할 프레임워크의 부가 요소 제작 (cont.)

◆ 애니메이션 구성

✓ 멤버 변수

- 현재의 프레임에서 그려줄 범위 정보를 담을 Rect형 멤버 변수
- 애니메이션의 초당 프레임(FPS:Frame Per Second) 정보를 담을 멤버 변수
- 애니메이션 시트의 개수를 담을 멤버 변수

```
public class SpriteAnimation extends GraphicObject {
    … …
    private Rect m_rect;          // 그려줄 사각 영역
    private int m_fps;            // 초당 프레임
    private int m_iFrames;        // 프레임 개수
    … …
}
```

5.4 게임 프레임워크

❖ 게임에 사용할 프레임워크의 부가 요소 제작 (cont.)

◆ 애니메이션 구성

✓ 멤버 변수

- 애니메이션이 얼마나 진행되었는지를 알려주는 멤버변수 (시간이 지남에 따라 값이 증가하고, 그려지는 이미지가 바뀌도록 함)
- 개별 프레임의 높이와 넓이를 저장할 멤버 변수

```
public class SpriteAnimation extends GraphicObject {
    … …
    private int m_currentFrame;  // 최근 프레임
    private int m_spriteWidth;
    private int m_spriteHeight;
    … …
}
```

5.4 게임 프레임워크

❖ 게임에 사용할 프레임워크의 부가 요소 제작 (cont.)

◆ 애니메이션 구성

✓ 애니메이션 클래스 구현

```
public class SpriteAnimation extends GraphicObject  {
    … …
    public SpriteAnimation(Bitmap bitmap)  {
            super (bitmap);
            // 멤버 변수 초기화
            m_rect = new Rect(0, 0, 0, 0);
            m_frameTimer = 0;
            m_currentFrame = 0;
    }
    … …
}
```

5.4 게임 프레임워크

❖ 게임에 사용할 프레임워크의 부가 요소 제작 (cont.)

◆ 애니메이션 구성

✓ 애니메이션 클래스 구현

- 스프라이트 애니메이션 정보를 대입하는 initSpriteData 메서드

```
public class SpriteAnimation extends GraphicObject  {
    … …
    public void initSpriteData( int _width, int _height, int _fps, int iFrame)  {
            // 기본 정보 설정
            m_spriteWidth = _width;
            m_spriteHeight = _height;
            m_rect.top = 0;          m_rect.bottom = m_spriteHeight;
            m_rect.left = 0;         m_rect.right = m_spriteWidth;
            m_fps = 1000 / _fps; // 밀리초 단위 프레임
            m_iFrames = iFrame;
    }
    … …
}
```

5.4 게임 프레임워크

❖ 게임에 사용할 프레임워크의 부가 요소 제작 (cont.)

◆ 애니메이션 구성

✓ 구현된 클래스의 인스턴스를 생성하고 초기화하는 코드 작성

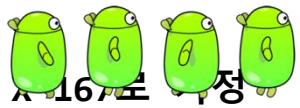
```
public class GameView extends SurfaceView implements SufaceHolder.Callback  {
    … …
    private SpriteAnimation m_spr;
    public GameView(Context context)  {
            super (context);
            getHolder( ).addCallback( this );
            m_thread = new GameViewThread(getHolder( ), this);
            m_spr = new SpriteAnimation(BitmapFactory.decodeResource
                    (getResource( ), R.drawable. walk));
    }
    … …
}
```

5.4 게임 프레임워크

❖ 게임에 사용할 프레임워크의 부가 요소 제작 (cont.)

◆ 애니메이션 구성

- ✓ SpriteAnimation 클래스의 InitSpriteData를 호출해서 이미지 파일에 대한 애니메이션 정보를 입력
- ✓ 사용할 이미지 : 좌우로 4개가 이어져 이미지 하나의 크기가 125 x 167로 구성



```
public class GameView extends SurfaceView implements SufaceHolder.Callback {
    … …
    public GameView(Context context) {
            super (context);
            getHolder( ).addCallback( this );
            m_thread = new GameViewThread(getHolder( ), this);
            m_spr = new SpriteAnimation(BitmapFactory.decodeResource
                    (getResource( ), R.drawable. walk));
            m_spr .initSpriteData(125, 167, 5, 4);
    }
    … …
}
```

5.4 게임 프레임워크

❖ 게임에 사용할 프레임워크의 부가 요소 제작 (cont.)

◆ 애니메이션 구성

✓ GameView 클래스의 onDraw 메서드에서 m_spr의 Draw 메서드를 호출

```
public class GameView extends SurfaceView implements SufaceHolder.Callback {
    … …
    @Override
    public void onDraw(Canvas canvas) {
            m_spr .Draw(canvas);
    … …
}
```

- SpriteAnimation 클래스의 Draw 메서드를 재정의하지 않았기 때문에 GraphicObject의 Draw 메서드를 호출하기 때문
- SpriteAnimation 클래스의 Draw 메서드를 재정의

5.4 게임 프레임워크

❖ 게임에 사용할 프레임워크의 부가 요소 제작 (cont.)

◆ 애니메이션 구성

✓ SpriteAnimation 클래스의 Draw 메서드 재정의

```
public class SpriteAnimation extends GraphicObject  {
    … …
    @Override
    public void Draw(Canvas canvas)  {
            Rect dest = new Rect( m_x, m_y, m_x + m_spriteWidth,
                                  m_y + m_spriteHeight);
            canvas.drawBitmap( m_bitmap, m_rect, dest, null);
    }
    … …
}
```

5. 4 게임 프레임워크

❖ 게임에 사용할 프레임워크의 부가 요소 제작 (cont.)

◆ 애니메이션 구성

✓ 시간이 지남에 따라 그려야 하는 프레임을 바꾸는 Update 메서드 추가

```
public class SpriteAnimation extends GraphicObject {
    … …
    public void Update( long gameTime) {

    }
    … …
}
```

로 게임 상의 시간을 받음

- 이를 이전에 그림을 그렸던 시간과 비교해서 다음 이미지를 그릴 시간이 되면 프레임을 변경해 주는 루틴을 추가

5.4 게임 프레임워크

❖ 게임에 사용할 프레임워크의 부가 요소 제작 (cont.)

◆ 애니메이션 구성

✓ Update 메서드 추가

```
public class SpriteAnimation extends GraphicObject  {
    … …
    public void Update( long gameTime)  {
            if ( gameTime > m_frameTimer + m_fps )  {
                m_frameTime = gameTime;
                m_currentFrame += 1;
            }
    }
    … …
}
```

5.4 게임 프레임워크

❖ 게임에 사용할 프레임워크의 부가 요소 제작 (cont.)

◆ 애니메이션 구성

✓ GameView 클래스에서 Update 메서드 호출

```
public class GameView extends SurfaceView implements SufaceHolder.Callback {
    … …
    public void Update( ) {
            long gameTime = System.currentTimeMillis( );
            m_spr .Update( gameTime );
    … …
}
```

- 움직임이 없다?? → 시간에 따라 Update를 해주어도 그려지는 프레임에 대한 값이 변하지 않음
- m_rect 변수의 값을 교체

5.4 게임 프레임워크

❖ 게임에 사용할 프레임워크의 부가 요소 제작 (cont.)

◆ 애니메이션 구성

✓ 프레임 교체

```
public class SpriteAnimation extends GraphicObject  {
    … …
    public void Update( long gameTime)  {
            if ( gameTime > m_frameTimer + m_fps )  {
                m_frameTime = gameTime;
                m_currentFrame += 1;
                // 프레임의 순환
                if ( m_currentFrame >= m_iFrames)       m_currentFrame = 0;
            }
            m_rect.left = m_currentFrame * m_spriteHeight;
            m_rect.right = m_rect.left + m_spriteWidth;
    }
    … …
}
```

5.4 게임 프레임워크

❖ 게임에 사용할 프레임워크의 부가 요소 제작 (cont.)

◆ 애니메이션 구성

- ✓ 애니메이션 실행 결과 확인
- ✓ 전체 코드

```
public class SpriteAnimation extends GraphicObject  {
    private Rect m_rect;
    private int m_fps;
    private int m_iFrames;
    private int m_currentFrame;
    private long m_frameTimer;
    private int m_spriteWidth;
    private int m_spriteHeight;
```

5. 4 게임 프레임워크

```
public SpriteAnimation(Bitmap bitmap)  {
        super (bitmap);
        m_rect = new Rect(0, 0, 0, 0);
        m_frameTimer = 0;
        m_currentFrame = 0;
}

public void initSpriteData( int _width, int _height, int _fps, int iFrame)  {
        m_spriteWidth = _width;
        m_spriteHeight = _height;
        m_rect.top = 0;         m_rect.bottom = m_spriteHeight;
        m_rect.left = 0;        m_rect.right = m_spriteWidth;
        m_fps = 1000 / _fps; // 밀리초 단위 프레임
        m_iFrames = iFrame;
}
```

5. 4 게임 프레임워크

```
    @Override
    public void Draw(Canvas canvas)  {
            Rect dest = new Rect( m_x, m_y, m_x + m_spriteWidth,
                                  m_y + m_spriteHeight);
            canvas.drawBitmap( m_bitmap, m_rect, dest, null);
    }

    public void Update( long gameTime)  {
            if ( gameTime > m_frameTimer + m_fps )  {
                m_frameTime = gameTime;
                m_currentFrame += 1;
                if ( m_currentFrame >= m_iFrames)       m_currentFrame = 0;
            }
            m_rect.left = m_currentFrame * m_spriteHeight;
            m_rect.right = m_rect.left + m_spriteWidth;
    }
}
```

5.4 게임 프레임워크

❖ 프레임 워크 마무리

◆ 구축된 프레임 워크

- ✓ SurfaceView를 통한 빠른 그래픽 처리 기반
- ✓ 애플리케이션을 관리하는 AppManager
- ✓ SoundPool 기반의 SoundManager
- ✓ State 패턴을 이용한 상태 기반
- ✓ 그래픽을 쉽게 그리기 위한 GraphicObject
- ✓ 게임에 필수인 SpriteAnimation

5.4 게임 프레임워크

❖ 프레임 워크 마무리 (cont.)

◆ 화면 상단의 타이틀 바와 안테나 탭 제거

✓ AndroidManifest.xml 파일 (text mode)

- android:theme 설정

✓ Java code

- Activity의 onCreate 메서드에 코드 추가

```
// 최상단 탭과 타이블 바를 제거하여 전체 화면으로 만들어줌
requestWindowFeature( Window. FEATURE_NO_TITLE);
GetWindow( ).setFlags( WindowManager.LayoutParams. FLAG_FULLSCREEN,
                       WindowManager.LayoutParams. FLAG_FULLSCREEN);
```

5.4 게임 프레임워크

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
                                    package="org.Framework"
                                    android:versionCode="1"
                                    android:versionName="1.0“>
<application android:icon="@drawable/icon" android:label="@string/app_name" >
    <activity android:name=".GameActivity"
            android:label="@string/app_name"
            android:theme="@android:style/Theme.Black.NoTitleBar.Fullscreen">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
    </activity>
</application>
<uses-sdk android:minSdkVersion="8" />
</manifest>
```