09 – Audio, Pause and Alert Dialog

Prepared by Tan Siew Lan, Tan_Siew_Lan@nyp.edu.sg & Credits to SlidesCarnival icons

Sounds as Background music (To be Completed - 1)

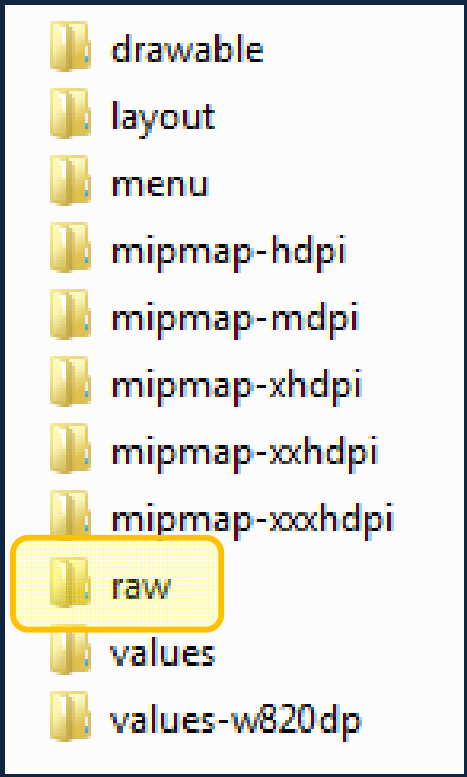
Prepared by Tan Siew Lan, Tan_Siew_Lan@nyp.edu.sg & Credits to SlidesCarnival icons

Audio

- Media can be video or audio
- Supports most of the common audio format
  - MP3(.mp3), WAVE(.wav), MIDI(.mid)
- Can be played from a raw resource, a file from the system or even a URL

Audio

- Place the sound or the audio clip into the **res/raw** folder



Step 1 – Background Music

- http://developer.android.com/reference/android/media/MediaPlayer.html
- Use of Media player class.
- *Note: Remember to import any Library needed*
- Define a **Media Player** in the **GamePanelSurfaceView** class

```
MediaPlayer bgm;
```

Step 2 – Background Music

- Load audio file (Hint: Under **GamePanelSurfaceView**'s **constructor**)

```
// Background music
bgm = MediaPlayer.create(context, R.raw.background_music);
```

- In **surfaceCreated()** method, play the audio file and set volume
  - **bgm.setVolume (float Left vol, float Right vol)**

```
bgm.setVolume(0.8f, 0.8f);
bgm.start();
```

Step 3 – Background Music

- End the audio file upon ending the thread/ end of app (Hint: Under the **surfaceDestroyed ()** method) and also release the memory used

```
//End background music
bgm.stop();
bgm.release();
```

- Practical Submission
  - Implement a background music to your scene

Sounds as Background music

Extra Bonus
(To be Completed - 2)

Prepared by Tan Siew Lan, Tan_Siew_Lan@nyp.edu.sg & Credits to SlidesCarnival icons

Extra Bonus

- Implement adjustment of volume of background music
- Set buttons to decrease and increase volume using **setVolume()**

Sounds as Feedback
(To be Completed - 3)

Prepared by Tan Siew Lan, Tan_Siew_Lan@nyp.edu.sg & Credits to SlidesCarnival icons

Step 1 – Feedback sounds

- http://developer.android.com/reference/android/media/SoundPool.html
- Use of **SoundPool** Class
- Plays audio that does not last longer than a few seconds
- Makes playing back sound effects easy
- Able to define various number of sound effects and play simultaneously

Step 2 – Feedback sounds

- Define a new **SoundPool object** and also define **sound IDs for each audio** to be played (Example: 2 sound effects, hit and explosion)

```
// Sound
private SoundPool sounds;
private int soundcorrect, soundwrong, soundbonus;
```

Step 3 – Feedback sounds

- Define how the SoundPool will be used (Hint: Under **GamePanelSurfaceView**'s constructor)

- Example:

  sounds = new SoundPool(10, AudioManager.*STREAM_MUSIC,0);*

  - 1st parameter – Max number of sound effects
  - 2nd parameter – Stream type used (currently is music stream where volume controls can be used and also normally used for games)
  - 3rd parameter – Sample-rate converter quality (0 is default where no effect is used)

```
//Define Soundpool will be used
sounds = new SoundPool(2, AudioManager.STREAM_MUSIC,0);
```

Step 4 – Feedback sounds

- Load the audio file from specified (Hint: Under **GamePanelSurfaceView**'s constructor)

- Example:

  hitsound = sounds.load(context, R.raw.*hit, 1);*
  exposound = sounds.load(context, R.raw.*explosion, 1);*

  - 3rd parameter – priority of sound (Normally set to 1)

```
//Load the audio file from specified
soundcorrect = sounds.load(context, R.raw.correct, 1);
soundwrong = sounds.load(context, R.raw.incorrect, 1);
```

Step 5 – Feedback sounds

- Play the audio file
- Example:

  sounds.play(hitsound, 1.0f, 1.0f, 0, 0, 1.5f);
  - 1st parameter – SoundID
  - 2nd parameter – left volume value (range 0.0 to 1.0)
  - 3rd parameter – right volume value (range 0.0 to 1.0)
  - 4th parameter – stream priority (0 = lowest priority)
  - 5th parameter – loop mode(0 = no loop, -1 = loop forever)
  - 6th parameter – playback rate (1.0 = normal, range 0.5 to 2.0)

```
sounds.play(soundcorrect, 1.0f, 1.0f, 0, 0, 1.5f);
```

Step 6 – Feedback sounds

- End the audio file upon ending the thread/ end of app (Hint: Under the **surfaceDestroyed () method**) and also release the memory used

```
//End the audio file upon ending
sounds.unload(soundcorrect);
sounds.unload(soundwrong);
sounds.release();
```

- Practical Submission
  - Implement sound effects upon any collision between objects in your scene

Pause buttons

(To be Completed - 4)

Prepared by Tan Siew Lan, Tan_Siew_Lan@nyp.edu.sg & Credits to SlidesCarnival icons

Step 1 – Pause

- On pause button pressed, game thread have to hold and wait

- Define a flag to pause the game state and load 2 button images

```
// Pause button state
private boolean pausepress = true;
private Objects PauseB1;
private Objects PauseB2;
```

Step 2 – Pause

- Create a new class, **Object.class**
- It takes in an image which is bitmap and the size of the image
- Add codes from the given txt file

Step 3 – Pause

- Load the 2 images for the Pause button (Hint: Under **GamePanelSurfaceView**'s constructor)

```
// Load Pause button images
PauseB1 = new Objects(BitmapFactory.decodeResource(getResources(),R.drawable.pause), 72, 72);
```

Step 4 – Pause

- Create method, RenderPause (Canvas canvas)
- In the RenderPause method, check if button is pressed.
- Image of Pause button will change once it is pressed or resumed.

```
public void RenderPause(Canvas canvas){
// Draw Pause button
    canvas.drawBitmap(PauseB1.getBitmap(), PauseB1.getX(), PauseB1.getY(), null);

    if (pausepress == true){
        canvas.drawBitmap(PauseB2.getBitmap(), PauseB2.getX(), PauseB2.getY(), null);

    }
    else {
        canvas.drawBitmap(PauseB1.getBitmap(), PauseB1.getX(), PauseB1.getY(), null);
        pausepress = false;
    }
}
```

Step 5 – Pause

- On the RenderGameplay(Canvas canvas) method, run RenderPause(canvas);

- Practical Submission
  - Implement Pause to your scene

  - Use onTouchEvent
  - When not moving ship, hence moveShip = false;
  - Check Collision of your touch on screen via to the image pause button
  - If **Not** pausepress, pausepress = true
    myThread.pause();
    bgm.pause();
  - Else pausepress = false
    myThread.unpause();
    bgm.start();

Alert Dialog

(To be Completed - 5)

Prepared by Tan Siew Lan, Tan_Siew_Lan@nyp.edu.sg & Credits to SlidesCarnival icons

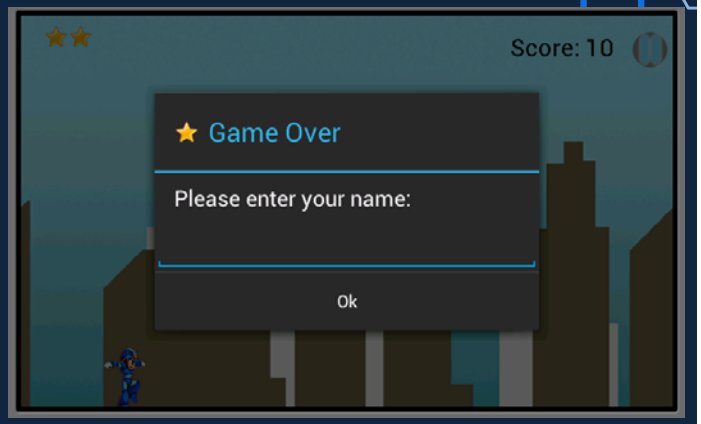
Alert Dialog

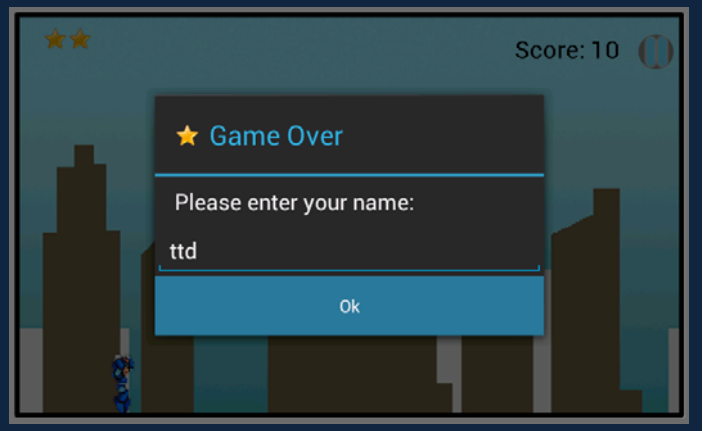
- A small pop up window that appears in front of the current activity
- Used for notifications
- **Alert Dialog**
  - Most common used dialog user interface
  - Features allows
    - Title
    - Text Message
    - 1, 2, or 3 buttons
    - Selectable items
    - (optional checkboxes or radio buttons)

- http://developer.android.com/guide/topics/ui/dialogs.html#AlertDialog

Alert Dialog

- In this example, it will illustrate
  - When it is the end game, there will be a pop up Alert
  - It prompts player to enter their name
  - After player enters name, press OK
  - It will returns to the Main Menu





Step 1 – Alert Dialog

- Append to the list of import statements

```
import android.content.DialogInterface;
import android.app.Activity;
import android.app.AlertDialog;
import android.content.Context;
import android.content.Intent;
import android.widget.EditText;
import android.text.InputFilter;
import android.text.InputType;
```

Step 2 – Alert Dialog

- Create a new class, **Alert.Class**

```java
import android.os.Handler;
import android.os.Looper;

public class Alert {

    private GamePanelSurfaceView Game;
    public Alert(GamePanelSurfaceView Game)
    {
        this.Game = Game;
    }

    public void RunAlert() {

        Handler handler = new Handler(Looper.getMainLooper());

        handler.postDelayed(new Runnable() {

            @Override
            public void run()
            {
                Game.alert.show();
            }
        }, 1000 );
    }
}
```

Step 3 – Alert Dialog

- In the **GamePanelSurfaceView** class,
  - Use AlertDialog Object
  - Use Activity tracker is used to track and then launch the desired activity

```
//  Alert  //
public boolean showAlert = false;
AlertDialog.Builder alert = null;
Activity activityTracker;
public boolean showed = false;
private Alert AlertObj;
```

Step 4 – Alert Dialog

- As we are using the activity tracker, constructor in my GamePanelSurfaceView class has to be changed

```
//constructor for this GamePanelSurfaceView class
public GamePanelSurfaceView (Context context, Activity activity){

    // Context is the current state of the application/object
    super(context);

    // To track an activity
    activityTracker = activity;

    // Adding the callback (this) to the surface holder to intercept events
    getHolder().addCallback(this);
```

Step 5 – Alert Dialog

- And in the **Gamepage** activity,

```
public class GamePage extends Activity{
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        requestWindowFeature(Window.FEATURE_NO_TITLE);
        // Making it full screen window
        getWindow().setFlags(WindowManager.LayoutParams.FLAG_FULLSCREEN, WindowManager.LayoutParams.F
        // Set our GamePanelSurfaceView as the View
        setContentView(new GamePanelSurfaceView(this, this));
    }
```

Step 6 – Alert Dialog

- Under **GamePanelSurfaceView**'s constructor, add codes

```
//Create Alert Dialog
AlertObj = new Alert(this);
alert = new AlertDialog.Builder(getContext());

//Allow players to input their name
final EditText input = new EditText(getContext());

//Define the input method where 'enter' key is disabled
input.setInputType(InputType.TYPE_CLASS_TEXT);

//Define max of 20 characters to be entered for 'Name' field
int maxLength = 20;
InputFilter[] FilterArray = new InputFilter[1];
FilterArray[0] = new InputFilter.LengthFilter(maxLength);
input.setFilters(FilterArray);
```

Step 7 – Alert Dialog

- Under **GamePanelSurfaceView**'s constructor, add codes

```
//Setup the alert dialog
alert.setCancelable(false);
alert.setView(input);

alert.setPositiveButton("Ok", new DialogInterface.OnClickListener()
{
    // do something when the button is clicked
    public void onClick(DialogInterface arg0, int arg1) {
        Intent intent = new Intent();
        intent.setClass(getContext(), Mainmenu.class);
        activityTracker.startActivity(intent);
    }

});
```

Step 8 – Alert Dialog

- Under **Update() method,** add codes

```
if (showAlert == true && !showed)
{
    showed = true;
    alert.setMessage("GameOver");
    AlertObj.RunAlert();
    showAlert = false;
    showed = false;
}
```

Step 9 – Alert Dialog

- Practical Submission
  - Implement pop up alert to your scene

```
if (hits == 0)
{
    //EndLevel();
    showAlert = true;
}
```

- Bonus
  - Use an activity tracker to transit from Game scene to Main menu upon GameOver

End