Có rất nhiều engine lập trình game di động như

**Unity 3D** <http://unity3d.com>

**Cocos2d-x** <http://www.cocos2d-x.org><http://cocos2d-x.vn/>

**Corona SDK**  <http://coronalabs.com/>

**Game Salad** <http://gamesalad.com/>

**Libgdx**  <http://libgdx.badlogicgames.com/index.html>

**Andengine**  <http://www.andengine.org/>

**Sprite Kit**  <http://www.sprite-kit.com/>

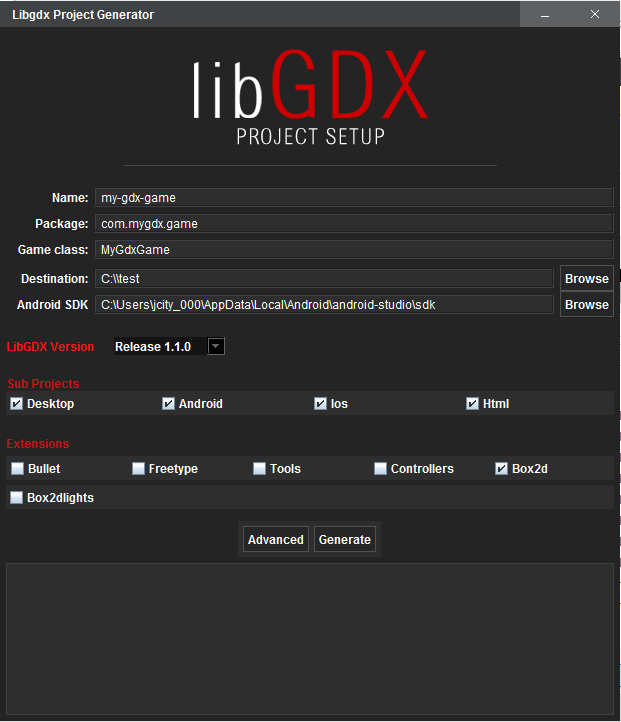
**Gideros** <http://giderosmobile.com/>

Công cụ được tôi chọn trong số này là LIBGDX

Bài 1: thiết lập Android Studio để lập trình game

Thiết lập môi trường cho Android Studio

Chạy file gdx-setup.jar tại <https://bitly.com/1i3C7i3>



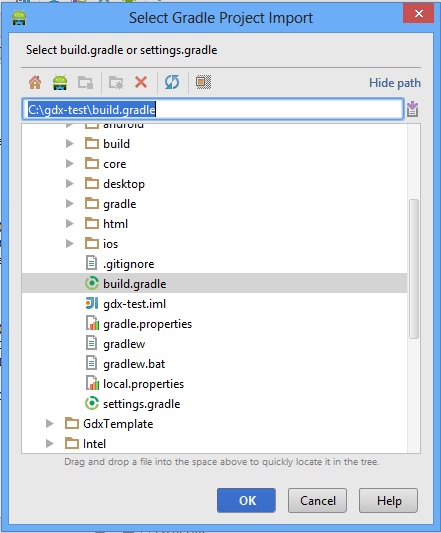
**Bạn cần thiết lập đường dẫn tới thư mực cài Android SDK. Có thể tìm thấy đường dẫn đến Android SDK bằng cách trong Android Studio, vào** Tools -> Android -> SDK Manage.

Có thể bỏ chọn IOS, Html, Desktop nếu ta chỉ phát triển game cho Android.

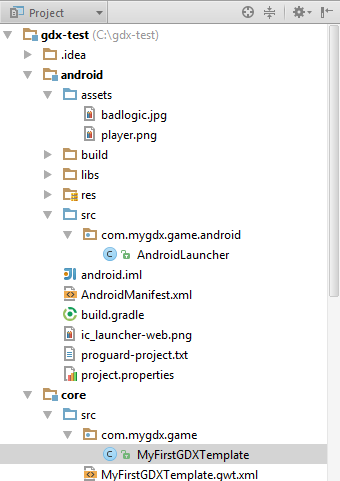
Khi đã thiết lập xong, chọn generate để tạo dự án

Import dự án vào Android Studio

Sau khi gdx-setup đã chạy xong, import tệp build.gradle vào Android Studio bằng cách vào File -> Import và tìm đến file build.gradle trong thư mục của dự án mà gdx-setup tự generate



**Click OK. Sau khi Android Studio Import xong, cấu trúc thư mục dự án như sau:**



**Biên dịch và chạy thử:**

**Sau khi chạy thử, ta sẽ thấy màn hình chuyển sang màu đỏ và một ảnh ví dụ tên gadlogic.jpg hiển thị ở góc dưới cùng bên trái màn hình thiết bi.**

**Kết luận:**

**GDX Lib là một thư viện java linh hoạt cho phép tạo ra các game độc lập thiết bị. để sử dụng trong môi trường Android, ta để gdx-setup tự generate dự án.**

**Bài tiếp theo ta sẽ nghiên cứu cách di chuyển đối tượng xung quanh màn hình và bắt đầu hành trình tạo một game đầu tiên**

# Phần II: Lập trình game trong Android sử dụng GDX lib

**Giới thiệu:**

Trong Phần I, chúng ta đã biết cách thiết lập thư viện GDX để bắt đầu lập trình game. Trong phần này, chúng ta sẽ tìm hiểu về cấu trúc, cách tạo một hiệu ứng đơn giản, phản hồi lại một thao tác chạm và phát âm thanh trong game. Khi hoàn thành bài này, chúng ta sẽ xây dựng được một game phát ra một tiếng nổ khi chạm ngón tay vào màn hình thiết bị.

**Cấu trúc game**

Khi tạo một template game sử dụng thư viện gdx trong phần 1, Android gdx đã tạo một class cho phép chúng ta viết code để thực hiện các chức năng của ứng dụng. Lớp này extend từ lớp cơ sở ApplicationAdapter, và ta có thể override các hàm create và render để tạo một bitmap và thể hiện nó lên màn hình. Hầu hết code này đã được tự tạo ra cho bạn. Method create của ApplicationAdapter là chỗ để ta viết code vẽ thành phần media của game. Method render được gọi mỗi 0.025 giây bởi vòng lặp. Trong method render chúng ta có thể di chuyển các ảnh, cập nhật hoạt cảnh và các chức năng hiển thị màn hình khác. Lớp Texture cho phép load một file ảnh từ thư mục assets và thể hiện lên màn hình. Chúng ta tạo texture để sử dụng bên trong method create, và vẽ hình rong method render. Nói chung, nên đặt tất cả image sẽ thể hiện trong một khu vực gọi là một SpriteBatch. SpriteBatch gộp tất cả các image để thể hiện và vẽ chúng một lần. Cách này rất hiệu quả để render các image trong game hơn là render độc lập

img = **new** Texture(**"player.png"**); *// the texture is loaded from the player.png file in the assets folder* }  
@Override  
**public void** render () {  
  
 Gdx.gl.glClearColor(1, 0, 0, 1);  
 Gdx.gl.glClear(GL20.GL\_COLOR\_BUFFER\_BIT);  
  
 batch.begin();  
 batch.draw(img, 100, 100);  
 batch.end();  
 }  
 }

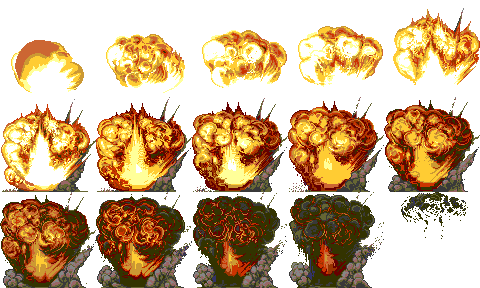
**Điều khiển thao tác chạm**

Trong nhiều game, thiết bị phản hồi lại thao tác chạm khi bạn đặt ngón tay trên thiết bị. thư viện GDX cung cấp phương pháp tương tác với thao tác chạm thông qua một interface tên là InputProcessor. Khi chsugn ta thực thi (implement) InputProcessor trong lớp game, Android studio nhắc ta phải generate tất cả các method của interface để tương tác với thao tác nhập. Đoạn code sau được generate cho interface InputProcessor.

**package** com.mygdx.game;  
  
 **import** com.badlogic.gdx.ApplicationAdapter;  
 **import** com.badlogic.gdx.Audio;  
 **import** com.badlogic.gdx.Gdx;  
 **import** com.badlogic.gdx.InputProcessor;  
 **import** com.badlogic.gdx.audio.Sound;  
 **import** com.badlogic.gdx.graphics.GL20;  
 **import** com.badlogic.gdx.graphics.OrthographicCamera;  
 **import** com.badlogic.gdx.graphics.Texture;  
 **import** com.badlogic.gdx.graphics.g2d.Animation;  
 **import** com.badlogic.gdx.graphics.g2d.BitmapFont;  
 **import** com.badlogic.gdx.graphics.g2d.SpriteBatch;  
 **import** com.badlogic.gdx.graphics.g2d.TextureRegion;  
 **import** com.badlogic.gdx.math.Vector3;  
  
**public class** MyFirstGDXTemplate **extends** ApplicationAdapter **implements** InputProcessor {  
  
 **public void** create () {  
 batch = **new** SpriteBatch();  
 img = **new** Texture(**"player.png"**);  
  
 }  
  
 @Override  
 **public void** render () {  
  
 Gdx.gl.glClearColor(1, 0, 0, 1);  
 Gdx.gl.glClear(GL20.GL\_COLOR\_BUFFER\_BIT);  
  
 batch.begin();  
 batch.draw(img, 100, 100);  
  
 }  
  
 @Override  
 **public boolean** keyDown(**int** keycode) {  
 **return false**;  
 }  
  
 @Override  
 **public boolean** keyUp(**int** keycode) {  
 **return false**;  
 }  
  
 @Override  
 **public boolean** keyTyped(**char** character) {  
 **return false**;  
 }  
  
 @Override  
 **public boolean** touchDown(**int** screenX, **int** screenY, **int** pointer, **int** button) {  
  
 }  
  
 @Override  
 **public boolean** touchUp(**int** screenX, **int** screenY, **int** pointer, **int** button) {  
 **return false**;  
 }  
  
 @Override  
 **public boolean** touchDragged(**int** screenX, **int** screenY, **int** pointer) {  
 **return false**;  
 }  
  
 @Override  
 **public boolean** mouseMoved(**int** screenX, **int** screenY) {  
 **return false**;  
 }  
  
 @Override  
 **public boolean** scrolled(**int** amount) {  
 **return false**;  
 }  
}

Nếu ta chạy đoạn code này trên thiết bị android trong chế độ debug và đặt một breakpoint trong method touchDown đã được generate, chúng ta thấy chương trình dừng tại method này khi ta chạm vào màn hình. Chúng ta có thể đặt tọa độ chạm bên trong method này để dùng trong game.

**Tạo một Animation**

****

Các animation được tạo ra từ các image, bao gồm một chuỗi tuần tự animation. Lớp TextureRegion được dùng để lấy ra từng ảnh phụ của cảnh nổ theo thứ tự hoạt ảnh. Lớp này sẽ lấy mỗi ảnh bắt đầu từ góc trên cùng bên trái và di chuyển đến bên phải, sau đó di chuyển đến dòng tiếp theo đến khi gặp ảnh cuối cùng trong hàng và lấy ra tất cả 15 ảnh. Mỗi khi những ảnh này được phân tích, chúng có thể được thêm vào lớp Animation thực thi animation. Trong method create ta thêm code sau:

explosionSheet = **new** Texture(Gdx.files.internal(**"explosion.png"**)); *// explosion image sheet shown above* TextureRegion[][] textureRegions = TextureRegion.split(explosionSheet, explosionSheet.getWidth()/FRAME\_COLS, explosionSheet.getHeight()/FRAME\_ROWS); *// #10* explosionFrames = **new** TextureRegion[FRAME\_COLS \* FRAME\_ROWS];  
 **int** index = 0;  
 **for** (**int** i = 0; i < FRAME\_ROWS; i++) {  
 **for** (**int** j = 0; j < FRAME\_COLS; j++) {  
 explosionFrames[index++] = textureRegions[i][j];  
 }  
 }  
 explosionAnimation = **new** Animation(0.025f, explosionFrames);

Để thể hiện sự nổ trong method render, ta cần theo dõi tình trạng của animation. Khi muốn thể hiện một vụ nổ, mỗi lần gọi method render, ta cần tăng vị trí của animation để lấy ảnh tiếp theo. Chúng ta sử dụng method getDeltaTime để lấy giá trị tiếp theo của game và thêm nó vào stateTime hiện tại. Đoạn code dưới đây sẽ vẽ animation khi người dùng chạm vào màn hình. Chúng ta sẽ thảo luận kỹ hơn ở phần tiếp theo. Chú ý là explosionAnimation có một method getKeyFrame. Tham số đầu tiên của method này lấy stateTime hiện tại để quyết định xem ảnh nào trong thứ tự animation sẽ được render. Tham số thứ hai cho biết ta có lặp lại animation không. Trong ví dụ này chúng ta sẽ không lặp, chỉ thể hiện 15 ảnh theo thứ tự . Hãy để ý câu lệnh if trong đoạn bật explosionAnimation. Bên trong câu lệnh if, lớp Animation có một method tên là isAnimationFinished để cho phép chúng ta xác định khi nào animation thực hiện xong một lần.

stateTime += Gdx.graphics.getDeltaTime();  
 **if** (!explosionAnimation.isAnimationFinished(stateTime) && explosionHappening) {  
 currentFrame = explosionAnimation.getKeyFrame(stateTime, **false**); *// #16* camera.unproject(touchPoint.set(touchCoordinateX, touchCoordinateY, 0));  
 batch.draw(currentFrame, touchPoint.x, touchPoint.y);  
 }

**Thay đổi trạng thái sau một thao tác chạm**

Trong phần trước, chúng ta đã nói về interface InputProcessor. Điều ta muốn xảy ra ngay khi người dùng chạm vào màn hình là hoạt ảnh nổ xảy ra. Nói cách khác, tất cả 15 khung hình được chạy. trong method touchDown của Inputprocessor, chúng ta muốn tắt hiệu ứng nổ.

@Override  
**public boolean** touchDown(**int** screenX, **int** screenY, **int** pointer, **int** button) {  
 touchCoordinateX = screenX;  
 touchCoordinateY = screenY;  
 stateTime = 0;  
 explosionHappening = **true**;  
 **return true**;  
}

Method này lấy tọa độ chạm nơi ta muốn sự kiện nổ xảy ra. Đồng thời nó reset lại stateTime vì vậy tuần tự vụ nổ diễn ra bắt đầu từ đầu. Cuối cùng, method này đặt một cờ trong tiến trình của vụ nổ.

**Tọa độ của camera**

Tọa độ chạm trả về theo tọa độ màn hình, vì vậy ta cần dịch chúng sang tọa độ thực. Để làm điều này ta cần một camera. Trong ví dụ này, ta sử dụng camera để dịch tọa độ chạm để so khớp với vị trí ngón tay chạm trên màn hình vì vậy ta có thể vẽ animation tại vị trí thích hợp. Đầu tiên ta cần khởi tạo camera trong phương thức create như bên dưới:

camera = **new** OrthographicCamera(Gdx.graphics.getWidth(), Gdx.graphics.getHeight());  
 camera.position.set(camera.viewportWidth \* .5f, camera.viewportHeight \* .5f, 0f);  
  
<span style=**"font-family: Georgia, 'Times New Roman', 'Bitstream Charter', Times, serif; font-size: 13px; line-height: 19px;"**>To initialize the camera, we need to construct the camera with the width and height of the graphic area. Then we need to set the camera position to be smack in the middle of the screen. The camera uses 3D coordinates, but we are only concerned with the X and Y coordinates **for** the explosion, so we set the Z position to zero and the x and y coordinates of the camera to the screens midpoint.  
  
 Now that we have our camera created the way we want it, we can unproject the touch points in the render method to reflect the game world coordinates using a Vector3 **class** . The touchPoint, which is the instance of the Vector3 **class**, can then be used to set the animation location.  
  
</span>

camera.unproject(touchPoint.set(touchCoordinateX, touchCoordinateY, 0));  
 batch.draw(currentFrame, touchPoint.x, touchPoint.y);

**Tạo một trái bom!**

Vụ nổ cần có âm thanh đi kèm. Tạo âm thanh trong gdx lib rất dễ dàng. Trong method create chúng ta chỉ cần tạo một đối tượng Sound mới. Ta cần đặt một file âm thanh (boom.m4a) trong folder assets

sound = Gdx.audio.newSound(Gdx.files.internal(**"boom.m4a"**));

Trong method touch chúng ta chỉ cần bật âm thanh khi người dùng chạm vào màn hình :

@Override  
**public boolean** touchDown(**int** screenX, **int** screenY, **int** pointer, **int** button) {  
 sound.play();  
 touchCoordinateX = screenX;  
 touchCoordinateY = screenY;  
 stateTime = 0;  
 explosionHappening = **true**;  
 **return true**;  
}

**Kết luận**

**Để tạo một game, đầu tiên ta cần làm quen với các công cụ có trong thư viện gdx phục vụ cho việc tạo game. Ở phần II này chúng ta đã làm quen với việc vẽ các hình, tạo hoạt hình, phát hiện chạm, và bật âm thanh.**