ISEL / ADEETC

Master in Communication Networks and Multimedia Engineering

Interactive Multimedia Applications

Tutorial 4

Interactive Multimedia Applications

libGDX - 1st Part

(Hello World, Textures, Animations, Accelerometer and Events)

Rui Jesus

Introduction

This work aims to introduce a set of techniques commonly used in the development of games for Android using the game engine libGDX. The application developed in this tutorial includes animations with Sprites, the use of the accelerometer and events to capture the touch on the screen. Below is a set of links that you should consult during the development of this work.

Android Developers
http://developer.android.com/training/index.html

libGDX
https://libgdx.badlogicgames.com/index.html

libGDX Wiki
https://github.com/libgdx/libgdx/wiki

Box2D Manual
https://github.com/libgdx/libgdx/wiki/Box2d

Laboratory Work

libGDX Projet - 1

1. Download the `gdx-setup.jar` file to generate a libGDX project for Android. This file is an executable that allows you to launch a user interface to configure and generate a libGDX project. Run the file. To run on the command line do:

```
java -jar gdx-setup.jar
```

2. In the user interface (Figure 1) specify the application name, the package name, the name of the main class, the directory where to place the project (you must create and select a directory within the Android Studio workspace) and indicate the location of the Android SDK. Next, choose the platforms on which the project can run. In this first example select only Android. Finally, select the extension "Box2d" (the only one we will use in this tutorial) and generate the project.

Figure 1. User interface to generate libGDX project.

3. Open the project in Android Studio. The project consists of 3 parts: **android**, **core** and **gradle**. The **android** part consists of the usual components in an Android project. The gradle part has one more "build.gradle" file for the **core** module than usual. It is in the **core** part that we will develop our application. Below is the code that lets you create the libGDX application within the Android project. Run the program.

```
public class AndroidLauncher extends AndroidApplication {
	@Override
	protected void onCreate (Bundle savedInstanceState) {
		super.onCreate(savedInstanceState);
		AndroidApplicationConfiguration config = new
AndroidApplicationConfiguration();
		initialize(new MyGdxGame(), config);
	}
}
```

4. A libGDX application also has a well-defined lifecycle, which allows you to manage the various states of an application, such as creating the application, stopping, resuming, rendering, and terminating ("disposing"). This life cycle is implemented by the "ApplicationListener" interface. Below are the methods that allow you to manage the life cycle.

```
// Method called once when the application is created
@Override
public void create() {

}

// Method called by the game loop from the application every
// time rendering should be performed. Game logic updates
// are usually also performed in this method.
@Override
public void render() {
}

// This method is called every time the game screen is re-sized
// and the game is not in the paused state. It is also called
// once just after the create() method.
@Override
public void resize(int width, int height) {
}

// On Android this method is called when the Home button is
// pressed or an incoming call
// is received. A good place to save the game state.
@Override
public void pause() {
}

// This method is only called on Android, when the application
// resumes from a paused state.
@Override
public void resume() {
}

// Called when the application is destroyed. It is preceded by
// a call to pause()
@Override
public void dispose() {
}
```

5. Figure 2 shows the life cycle diagram of a **libGDX** application.

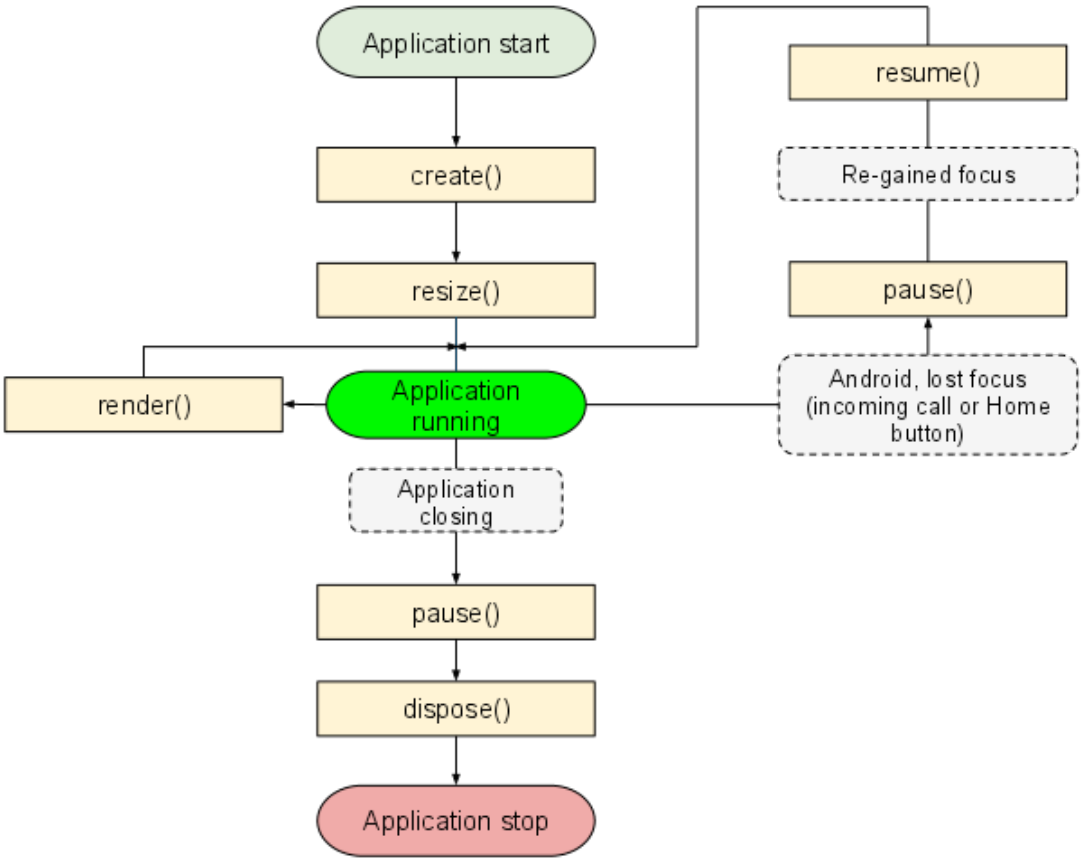

Figure 2. **LibGDX** application lifecycle.

6. **LibGDX** consists of 5 main interfaces that provide means for interacting with the operating system:

   - **Application**: executes the application and informs the client API of the events that occurred. It provides "login" facilities and methods for questioning the system state, e.g., memory usage.

```
// Android Version
int androidVersion = Gdx.app.getVersion();

// memory consumption
long javaHeap = Gdx.app.getJavaHeap();
long nativeHeap = Gdx.app.getNativeHeap();
```

- **Files**: allows you to access the platform's file system. Provides an abstraction about the different file locations.

```
// Image file in the assets directory
Texture myTexture = new Texture(Gdx.files.internal(“assets/
texture/brick.png”));
```

- **Input**: informs the client API of the user input, for example, through mouse, keyboard, touch screen or accelerometer events.

```
//  current touch coordinates if a touch event is in progress
if (Gdx.input.isTouched()) {
      System.out.println("Input occurred at x=" +
Gdx.input.getX() + ", y=" + Gdx.input.getY());
}
```

- **Net**: provides means to access resources via HTTP / HTTPS, and to create TCP servers and clients with "**sockets**".

- **Audio**: provides media for reproducing sounds, effects and streaming music and for accessing audio input / output devices ".

```
// plays a sound file from disk repeatedly with the volume half
turned up
Music music = Gdx.audio.newMusic(Gdx.files.getFileHandle("data/
myMusicFile.mp3", Files.FileType.Internal));
music.setVolume(0.5f);
music.play();
```

- **Graphics**: provides OpenGL ES 2.0 and allows you to query video configuration parameters and other similar media.

```
// clears the screen and paints it with red
Gdx.gl.glClearColor(1, 0, 0, 1);
Gdx.gl.glClear(GL20.GL_COLOR_BUFFER_BIT);
```

7. Change the background color and position the image in the center of the screen. Change the setting so that the application is executed in portrait mode. Run the program.

Hello libGDX - 1

8. To place text declare a "BitmapFont" object. In the "create" and "render" methods add the following lines of code. Run the program. Do not forget to call the "font.dispose ()" method inside the "dispose" callback.

```
// create
font = new BitmapFont();
font.setColor(Color.RED);
font.getData().setScale(3.0f, 3.0f);

// render
batch.begin();
batch.draw(img, Gdx.graphics.getWidth() / 2 - img.getWidth() / 2,
Gdx.graphics.getHeight() / 2 - img.getHeight() / 2);
      font.draw(batch, "Hello libGDX - Tutorial 4", 0,
Gdx.graphics.getHeight());
      batch.end();
```

9. Change the picture by putting in the same place the image "plain.png" provided by the teacher.

Dynamic Textures - 1

10. Textures may not come from files. To create new textures we will use the object "Pixmap". In the "create" method add the following lines of code. Run the program.

```
// create
//256 wide, 128 height using 8 bytes for Red, Green, Blue and Alpha
pixmap = new Pixmap(256,128, Pixmap.Format.RGBA8888);

pixmap.setColor(Color.RED);
pixmap.fill();

//Draw two lines forming an X
pixmap.setColor(Color.BLACK);
pixmap.drawLine(0, 0, pixmap.getWidth()-1, pixmap.getHeight()-1);
pixmap.drawLine(0, pixmap.getHeight()-1, pixmap.getWidth()-1, 0);

//Draw a circle about the middle
pixmap.setColor(Color.YELLOW);
pixmap.drawCircle(pixmap.getWidth()/2, pixmap.getHeight()/2,
pixmap.getHeight()/2 - 1);

cimage = new Texture(pixmap);
pixmap.dispose();
sprite = new Sprite(cimage);
```

11. In the "render" method, add the following lines of code. Run the program.

```
Gdx.gl.glClearColor(0, 0, 1, 1);
Gdx.gl.glClear(GL20.GL_COLOR_BUFFER_BIT);
batch.begin();
batch.draw(img, Gdx.graphics.getWidth() / 2 - img.getWidth() / 2,
Gdx.graphics.getHeight() / 2 - img.getHeight() / 2);
      font.draw(batch, "Hello libGDX - Tutorial 4", 0,
Gdx.graphics.getHeight());
      sprite.setPosition(0, Gdx.graphics.getHeight() / 2 -
sprite.getHeight()/2);
      sprite.draw(batch);
      sprite.setPosition(Gdx.graphics.getWidth()- sprite.getWidth(),
Gdx.graphics.getHeight() / 2 - sprite.getHeight()/2);
      sprite.draw(batch);
      batch.end();
```

Animations - 1

12. Copy the "spritesheet.atlas" and "spritesheet.png" files to the "assets" directory of the project. The second file combines several images into a single image and the first file is a kind of map (atlas) with the location of the various images that make up the second file. View the two files.

13. Let's build an animation with the various images placed in the "spritesheet.png" file. Add the code below for the "create" and "render" methods. Run the program. Do not forget to make "dispose" to the objects you have created.

```
// create
textureAtlas = new
TextureAtlas(Gdx.files.internal("spritesheet.atlas"));
animation = new Animation(1/15f, textureAtlas.getRegions());

// render
batch.begin();
…
elapsedTime += Gdx.graphics.getDeltaTime();
batch.draw(animation.getKeyFrame(elapsedTime, true), 0, 0);
batch.end();
```

14. With a "TextureAtlas" object we can make several animations with the same images in memory. Add the following code to make another animation. Place this animation in the lower right corner of the screen.

```
// create
…
TextureRegion[] rotateUpFrames = new TextureRegion[10];

rotateUpFrames[0] = (textureAtlas.findRegion("0001"));
rotateUpFrames[1] = (textureAtlas.findRegion("0002"));
rotateUpFrames[2] = (textureAtlas.findRegion("0003"));
rotateUpFrames[3] = (textureAtlas.findRegion("0004"));
rotateUpFrames[4] = (textureAtlas.findRegion("0005"));
rotateUpFrames[5] = (textureAtlas.findRegion("0006"));
rotateUpFrames[6] = (textureAtlas.findRegion("0007"));
rotateUpFrames[7] = (textureAtlas.findRegion("0008"));
rotateUpFrames[8] = (textureAtlas.findRegion("0009"));
rotateUpFrames[9] = (textureAtlas.findRegion("0010"));

rotateUpAnimation = new Animation(0.1f,rotateUpFrames);
```

AssetLoader - 2 (BUILD A NEW PROJECT)

15. Create a new **libGDX** project. Copy the following files to the "assets" directory of the project:

    "*face_box_tiled.png*"

    "*face_circle_tiled.png*"

    "*face_triangle_tiled.png*"

    "*face_hexagon_tiled.png*"

16. In general, in a game we have several "assets" to load into memory at the beginning. Create the "AssetLoader" class to accomplish this task, so that the application code gets more organized. The "AssetLoader" class should load the 4 images from point 15, construct "TextureRegion" objects with those textures and create their animations. Follow the example below for an image.

```
public class AssetLoader {
    public static Animation faceBoxAnimation,
faceCircleAnimation,faceHexAnimation,faceTriAnimation;
    public static TextureRegion box1, box2, circle1, circle2,
hex1,hex2,tri1,tri2;

    static Texture[] texture = new Texture[4];

    public static void load() {
        texture[0] = new
Texture(Gdx.files.internal("face_box_tiled.png"));


        box1 = new TextureRegion(texture[0], 0, 0, 32, 32);
        box2 = new TextureRegion(texture[0], 32, 0, 32, 32);

        TextureRegion[] boxs = { box1, box2 };
        faceBoxAnimation = new Animation(0.5f, boxs);

faceBoxAnimation.setPlayMode(Animation.PlayMode.LOOP_PINGPONG);

      }


     public static void dispose() {
        texture[0].dispose();
        texture[1].dispose();
        texture[2].dispose();
        texture[3].dispose();

    }
}
```

17. Add the line of code `AssetLoader.load()` in the callback method "create". Do the animation in the same way as in point 13 (in the "render" callback method). Run the program. Do not forget to "dispose" the objects you created.

User Input - Events - 2

18. In order to receive input from the user it is necessary to implement the "InputProcessor" interface. You must also register our "InputProcessor" with the global input instance (callback method "create") and redefine various callback methods. Copy the code below.

```
// create
Gdx.input.setInputProcessor(this);


// input callbacks
@Override
public boolean touchUp(int screenX, int screenY, int pointer,
int button) {

    return false;
}

@Override
public boolean touchDragged(int screenX, int screenY, int
pointer) {
    // TODO Auto-generated method stub
    return false;
}

@Override
public boolean mouseMoved(int screenX, int screenY) {
    // TODO Auto-generated method stub
    return false;
}

@Override
public boolean scrolled(int amount) {
    // TODO Auto-generated method stub
    return false;
}

@Override
public boolean keyDown(int keycode) {
    // TODO Auto-generated method stub
    return false;
}


@Override
public boolean keyUp(int keycode) {
    // TODO Auto-generated method stub
    return false;
}

@Override
public boolean keyTyped(char character) {
    // TODO Auto-generated method stub
    return false;
}

@Override
public boolean touchDown(int screenX, int screenY, int pointer,
int button) {
    // TODO Auto-generated method stub
```

19. Make a program that displays the `faceBoxAnimation` animation on the screen when the user touches the screen. In case the animation already being on the screen should be deleted. The animation should be rendered at the touch location.

Accelerometer - 2

20. To use the accelerometer we will create three attributes in the main class. In the "create" callback method you need to check if the device has the accelerometer available. And to read the acceleration values given by the accelerometer we will construct a private method. Use the code below.

```
// classe principal
boolean canUseAcel = false;
private float prevAccelX;
private float prevAccelY;

// create
…
canUseAcel =
Gdx.input.isPeripheralAvailable(Peripheral.Accelerometer);

// private method
// touchx and touchy are used in render callback to update the
// animation position
private void processAccelerometer() {
    float y = Gdx.input.getAccelerometerY();
    float x = Gdx.input.getAccelerometerX();
    if ((prevAccelX != x) || prevAccelY != y) {
        prevAccelX = x;
        prevAccelY = y;
        touchx += x;
        touchy += y;
    }
}
```

21. In order for the *faceBoxAnimation* animation used in point 19 to move on the screen according to the acceleration, change the "render" callback method to the code below. Run the program.

```
public void render () {
      Gdx.gl.glClearColor(0, 0, 1, 1);
      Gdx.gl.glClear(GL20.GL_COLOR_BUFFER_BIT);

       if(canUseAcel){
           processAccelerometer();
       }

      batch.begin();
       if (touch) {
           elapsedTime += Gdx.graphics.getDeltaTime();

batch.draw(AssetLoader.faceHexAnimation.getKeyFrame(elapsedTime,
true), touchx, touchy);
       }
       batch.end();
}
```

GameScreen Class - 2

22. In this tutorial we will only use a screen (screen) however, almost every game has several screens. So let's separate the code related to the display of elements on the screen of the main class (Box2DExample). And let's create the "GameScreen" class that will manage the application screen. Copy everything you have in the main class to the "GameScreen" class. Change the main class to the code below.

```
// Game implements ApplicationListener that delegates to a screen
// This allows an application to easily have multiple screens
// ApplicationAdapter class also implements ApplicationListener
public class box2DExample extends Game {

      @Override
      public void create() {
        AssetLoader.load();
        setScreen(new GameScreen());
    }

      @Override
      public void dispose() {
            super.dispose();
        AssetLoader.dispose();
      }

}
```

23. The code below outlines the most relevant parts of the "GameScreen" class. All callback methods related to user input are now in the "GameScreen" class. This class implements "Screen" that represents one of several screens of a game. You need to add more "callback" methods to implement events commonly used on screens. Run the program. The result will be the same as in point 21 but with another code organization that allows for more flexibility in managing multiple application screens.

```
public class GameScreen implements Screen, InputProcessor {

    SpriteBatch batch;
    float elapsedTime = 0;
    int touchx = 0;
    int touchy = 0;
    boolean touch = false;

    boolean canUseAcel =
Gdx.input.isPeripheralAvailable(Input.Peripheral.Accelerometer);
    private float prevAccelX;
    private float prevAccelY;

    public GameScreen() {
        batch = new SpriteBatch();
        Gdx.input.setInputProcessor(this);
    }
    @Override
    public void render(float delta) {
        Gdx.gl.glClearColor(0, 0, 1, 1);
        Gdx.gl.glClear(GL20.GL_COLOR_BUFFER_BIT);

        if (canUseAcel) {
            processAccelerometer();
        }

        batch.begin();
        if (touch) {
            elapsedTime += Gdx.graphics.getDeltaTime();

batch.draw(AssetLoader.faceHexAnimation.getKeyFrame(elapsedTime,
true), touchx, touchy);
        }
        batch.end();
    }
    @Override
    public void dispose() {
        batch.dispose();
        AssetLoader.dispose();

    }

    private void processAccelerometer() {
        float y = Gdx.input.getAccelerometerY();
        float x = Gdx.input.getAccelerometerX();
        if ((prevAccelX != x) || prevAccelY != y) {
            prevAccelX = x;
            prevAccelY = y;
            touchx -= x;
            touchy -= y;
        }
    }
    @Override
    public boolean touchUp(int screenX, int screenY, int pointer, int
button) {
        if (touch) touch = false;
        else {
            touch = true;
        }
        touchx = screenX;
        touchy = Gdx.graphics.getHeight() - screenY;
        return false;
    }
```

Camera - 2

24. To facilitate the movements in the game world, we will use a camera object that will allow us greater flexibility in the way we manage the movements on the game screen (e.g., translations, rotations or zoom operations, ...). Use an object of class "OrthographicCamera". In the "GameScreen" class constructor, add the code below. Run the program. The result should be the same as the previous point.

```
// Game implements ApplicationListener that delegates to a screen
camera = new OrthographicCamera();
camera.viewportHeight = Gdx.graphics.getHeight();
camera.viewportWidth = Gdx.graphics.getWidth();

//Setting the camera's initial position to the bottom left of the
map. The camera's position is in the center of the camera
camera.position.set(camera.viewportWidth * .5f, camera.viewportHeight
* .5f, 0f);

//Update our camera
camera.update();

//Update the batch with our Camera's view and projection matrices
batch.setProjectionMatrix(camera.combined);
```

Box2D - World - 2

25. Box2D is a library of 2D physics (physics engine) of the most popular in 2D games and used in most programming languages. To boot the physics engine you need to do `Box2D.init()`. However, when the "World" object is built, this Box2D boot is executed automatically. So, to build the object that will simulate the game world (and start Box2D) just put the line below in the "GameScreen" constructor:

```
// First argument sets the horizontal and vertical gravity forces
// Second argument is a boolean value which tells the world if we
// want objects to sleep or not (to conserve CPU usage)
World world = new World(new Vector2(0, 0), true);
```

26. The next step consist of declare ("GameScreen" attribute) and build ("GameScreen" constructor) a "Box2DDebugRenderer" object. In "release" mode it is not used but in "debug" mode it is useful to test the application.

27. To update the world, add the code below in the "GameScreen" class. This code defines how much how long the world is updated (e.g., 50 or 60 frames / second).

```
// GameScreen Atributes
static final float BOX_STEP = 1 / 60f;
static final int BOX_VELOCITY_ITERATIONS = 6;
static final int BOX_POSITION_ITERATIONS = 2;

// The best place to call our step function is at the end of our
render() loop
world.step(BOX_STEP, BOX_VELOCITY_ITERATIONS,
BOX_POSITION_ITERATIONS);
```

28. Add the line of code below to render the world in "Debug" mode. It should be placed in the "render" callback method before the "stepping" of the "world" object is defined, otherwise synchronization problems may occur.

```
debugRenderer.render(world, camera.combined);
```

29. The world needs to be built. For now the world is a square that defines the limits of the screen, that is, the world consists of 4 static objects. Copy the code below to the end of the "GameScreen" builder. Run the program. To better understand the code below you should consult the Box2D manual.

```
// bottom
BodyDef groundBodyDef = new BodyDef();
groundBodyDef.position.set(new Vector2(0, 0));
Body groundBody = world.createBody(groundBodyDef);
PolygonShape groundBox = new PolygonShape();
groundBox.setAsBox(camera.viewportWidth, .5f);
groundBody.createFixture(groundBox, 0.0f);
//top
groundBodyDef = new BodyDef();
groundBodyDef.position.set(new Vector2(0, camera.viewportHeight));
groundBody = world.createBody(groundBodyDef);
groundBox = new PolygonShape();
groundBox.setAsBox(camera.viewportWidth, .5f);
groundBody.createFixture(groundBox, 0.0f);
//left
groundBodyDef = new BodyDef();
groundBodyDef.position.set(new Vector2(0, camera.viewportHeight));
groundBody = world.createBody(groundBodyDef);
groundBox = new PolygonShape();
groundBox.setAsBox(.5f, camera.viewportHeight);
groundBody.createFixture(groundBox, 0.0f);
//right
…

debugRenderer = new Box2DDebugRenderer();
Gdx.input.setInputProcessor(this);
```

30. To construct dynamic objects we will create instances of the class "body" and we will insert them in the world. We have 4 objects (animations) with different shapes ("Box", "Circle", Triangle, Hexagon). First, let's create in the "GameScreen" class an enumerated type and an array:

```
// GameScreen Atributes
Array<Body> bodies = new Array<Body>();

public enum FormType {
    Box, Circle, Triangle, Hexagon
}
```

31. To create the different bodies, copy the methods below into the "GameScreen" class. Make the method for the "body" with a triangle shape.

```
// GameScreen methods
private void createBoxBody(int screenX, int screenY) {
    BodyDef bodyDef = new BodyDef();
    bodyDef.type = BodyDef.BodyType.DynamicBody;
    bodyDef.position.set(screenX, screenY);
    Body body = world.createBody(bodyDef);
    PolygonShape Box = new PolygonShape();
    //hx the half-width
    //the half-height
    Box.setAsBox(16.0f, 16.0f);
    FixtureDef fixtureDef = new FixtureDef();
    fixtureDef.shape = Box;
    fixtureDef.density = 1.0f;
    fixtureDef.friction = 0.0f;
    fixtureDef.restitution = 1;
    body.createFixture(fixtureDef);
    body.setUserData(FormType.Box);
}

private void createCircleBody(World world2, int screenX, int
screenY) {
      BodyDef bodyDef = new BodyDef();
      bodyDef.type = BodyType.DynamicBody;
      bodyDef.position.set(screenX, screenY);
      Body body = world.createBody(bodyDef);
      CircleShape dynamicCircle = new CircleShape();
      dynamicCircle.setRadius(15.0f);
      FixtureDef fixtureDef = new FixtureDef();
      fixtureDef.shape = dynamicCircle;
      fixtureDef.density = 1.0f;
      fixtureDef.friction = 0.0f;
      fixtureDef.restitution = 1;
      body.createFixture(fixtureDef);
      body.setUserData(FormType.Circle);
}
```

```
// GameScreen methods
private void createHexBody(World world2, int screenX, int screenY) {
      BodyDef bodyDef = new BodyDef();
      bodyDef.type = BodyType.DynamicBody;
      bodyDef.position.set(screenX, screenY);
      Body body = world.createBody(bodyDef);
      PolygonShape hex = new PolygonShape();
      Vector2[] vertices = new Vector2[6];
      vertices[0] = new Vector2(0, 16.0f); //top
      vertices[1] = new Vector2(0, -16.0f); //bottom
      vertices[2] = new Vector2(-14.0f, 8.25f); //left top
      vertices[3] = new Vector2(-14.0f, -8.25f); // left bottom
      vertices[4] = new Vector2(14.0f,8.25f);
      vertices[5] = new Vector2(14.0f, 8.25f);
      hex.set(vertices);
      FixtureDef fixtureDef = new FixtureDef();
      fixtureDef.shape = hex;
      fixtureDef.density = 1.0f;
      fixtureDef.friction = 0.0f;
      fixtureDef.restitution = 1;
      body.createFixture(fixtureDef);
      body.setUserData(FormType.Hexagon);
}
```

32. Update the callback method code "touchUp" with the code below. At this stage we are just creating rectangular bodies.

```
// GameScreen methods
public boolean touchUp(int screenX, int screenY, int pointer, int
button) {

    int h = Gdx.graphics.getHeight();
    int w = Gdx.graphics.getWidth();
    Gdx.app.log("touchUp", "Height:" + h + "Width " + w);
    createBoxBody(screenX, Gdx.graphics.getHeight() - screenY);
    return false;
}
```

33. To update the "world" object with the accelerometer values, copy the code below to the method "`processAccelerometer`". Again, to better understand the code you should consult the Box2D manual.

```
// GameScreen methods
private void processAccelerometer() {

    float y = Gdx.input.getAccelerometerY();
    float x = Gdx.input.getAccelerometerX();
    if ((prevAccelX != x) || prevAccelY != y) {
        prevAccelX = x;
        prevAccelY = y;

        world.setGravity(new Vector2(y, -x));
   }
}
```

34. In the "render" callback method, we will go through all the bodies of the "world" object and render them (code below). Run the program.

```
// GameScreen methods
public void render(float delta) {
    elapsedTime += delta;

    if (canUseAcel) {
        processAccelerometer();
    }

    Gdx.gl.glClearColor(0, 0, 1, 1);
    Gdx.gl.glClear(GL20.GL_COLOR_BUFFER_BIT);


    debugRenderer.render(world, camera.combined);
    batch.begin();

    world.getBodies(bodies);
    for (Body body : bodies) {
        Object data = body.getUserData();
        if (data != null) {
            if (data == FormType.Box) {

batch.draw(AssetLoader.faceBoxAnimation.getKeyFrame(elapsedTime,
true), body.getPosition().x - 16, body.getPosition().y - 16);
            }
        }
    }
    batch.end();
    world.step(BOX_STEP, BOX_VELOCITY_ITERATIONS,
BOX_POSITION_ITERATIONS);
}
```

35. Does it work as you expected? In principle, the objects appear face up (flip required) and when they collide in the world they begin to be misaligned with the bodies (the rotation needs to be updated). So make the changes below in the code. Run the program.

```
// AssetLoader
public class AssetLoader {
    public static Animation faceBoxAnimation,
faceCircleAnimation,faceHexAnimation,faceTriAnimation;
    public static TextureRegion box1, box2, circle1, circle2,
hex1,hex2,tri1,tri2;

    static Texture[] texture = new Texture[4];

    public static void load() {
              texture[0] = new
Texture(Gdx.files.internal("face_box_tiled.png"));
        …
         box1 = new TextureRegion(texture[0], 0, 0, 32, 32);
        box1.flip(false, true);
        box2 = new TextureRegion(texture[0], 32, 0, 32, 32);
        box2.flip(false, true);

        TextureRegion[] boxs = { box1, box2 };
        …
```

```
// GameScreen – render
…
for (Body body : bodies) {

      Object data = body.getUserData();
      if (data != null) {

            if (data == FormType.Box) {


batcher.draw(AssetLoader.faceBoxAnimation.getKeyFrame(stateTime,
true),
                                 body.getPosition().x -
AssetLoader.box1.getRegionWidth() / 2,
                                 body.getPosition().y -
AssetLoader.box1.getRegionHeight() / 2,
                                 AssetLoader.box1.getRegionWidth() / 2,
                                 AssetLoader.box1.getRegionHeight() /
2,
                                 AssetLoader.box1.getRegionWidth(),
                                 AssetLoader.box1.getRegionHeight(),
1.0f, 1.0f,
                                 body.getAngle() *
MathUtils.radiansToDegrees);
            }
…
```

36. Change the code of the previous box so that any of the 4 shapes can be represented. Run the program. Switch to landscape mode.

37. To create a different object each time you touch the screen, update the code with the code below. Run the program.

```
// GameScreen – atribute
RandomXS128 random = new RandomXS128(123);

// GameScreen – method
// Should be called in touchUp callback method instead of
// createBoxBody
private void addNewBody(int screenX, int screenY) {
      int type = random.nextInt(4);
      if (type == 0) {
            createBoxBody(this.world, screenX, screenY);
      }
      if (type == 1) {
            createCircleBody(this.world, screenX, screenY);
      }
      if (type == 2) {
            createTriBody(this.world, screenX, screenY);
      }
      if (type == 3) {
            createHexBody(this.world, screenX, screenY);
      }
}
```