

COMP20050 Software Engineering Project 2

14. LibGDX details (Part 2)

Assoc. Prof. Pavel Gladyshev



UCD School of Computer Science.

Scoil na Ríomheolaíochta UCD.

LibGDX files

Accessing files via LibGDX

LibGDX can run on a variety of platforms.

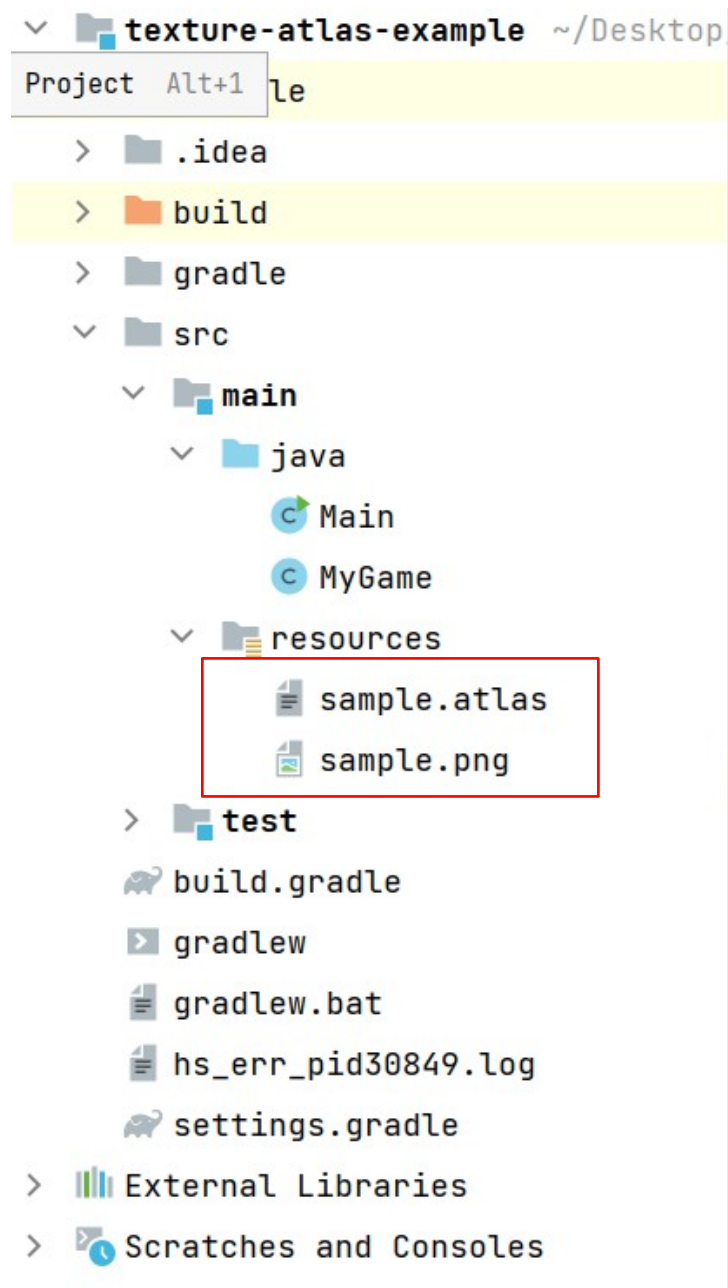
Gdx.files object provides uniform access to files on all different platforms supported by LibGDX.

Given that our project needs to read multimedia assets packaged in the Jar file, we will only touch upon Gdx.files functionality. For more detail please see <https://libgdx.com/wiki/file-handling>

- **Gdx.files** can access files stored in several different areas. Depending on the platform, some of these areas may or may not exist. In desktop environment, there are
 - Files packaged inside the Jar file (“**internal**” files), which can be accessed as read-only files.
 - Files stored in different areas of the computer file system: files in the user’s home directory (“**external**” files), files in the Java Classpath (“**classpath**” files), or arbitrary files anywhere in the file system (“**absolute**” files) references from the file system root.

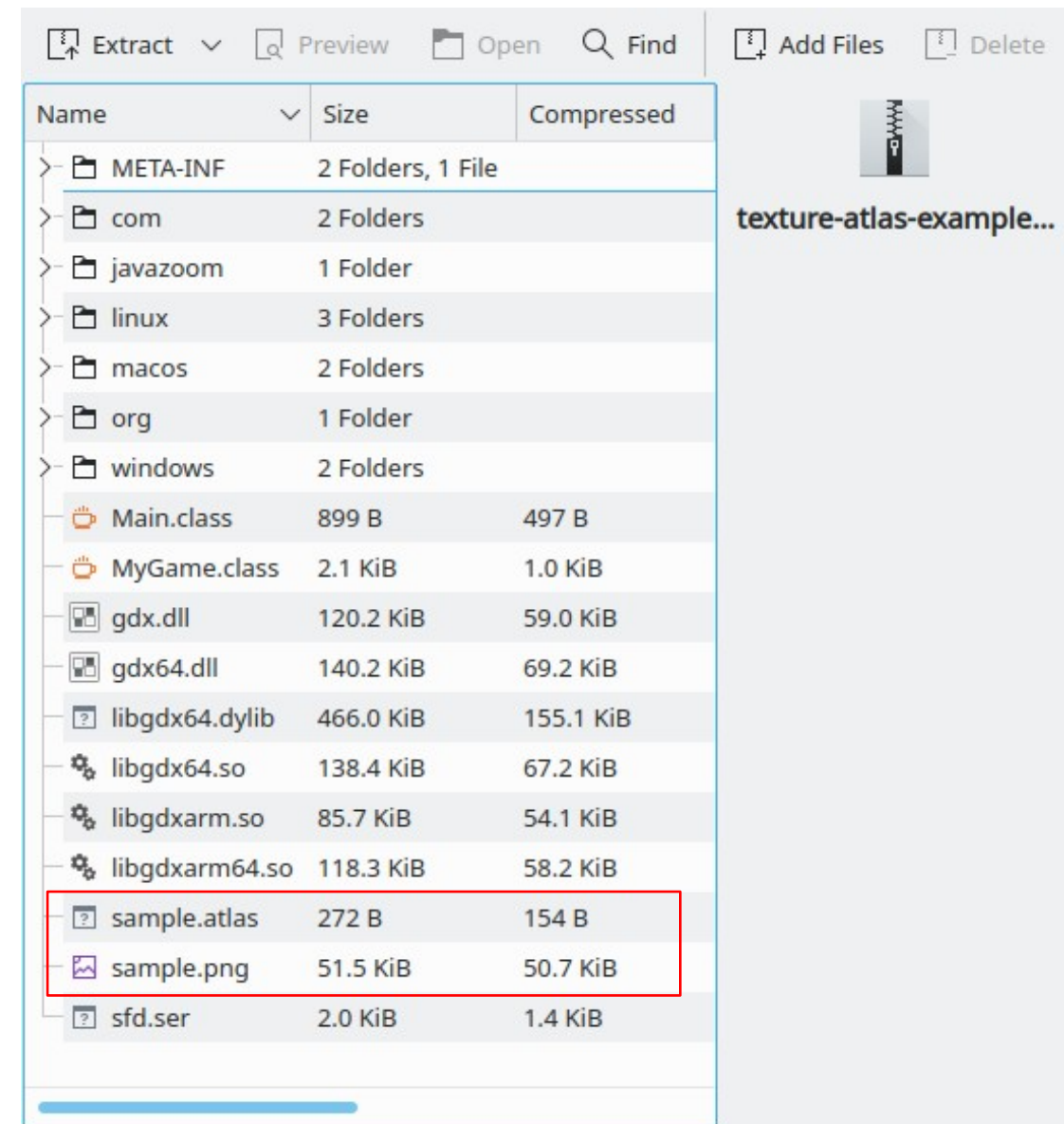
Creating and accessing internal files in the Jar

Gradle project:



copied by
gradle build
process

Generated Jar file



FileHandle h = Gdx.files.internal("sample.atlas");

Pathname here is relative to the root of Jar file

The contents of resources/ folder in src/main are copied into the root of Jar file by gradle

LibGDX Texture Atlas, Bitmap Font, Animation, Sprite,
Skin

Advanced graphics based on SpriteBatch functionality

The basic functionality provided by SpriteBatch can be easily extended:

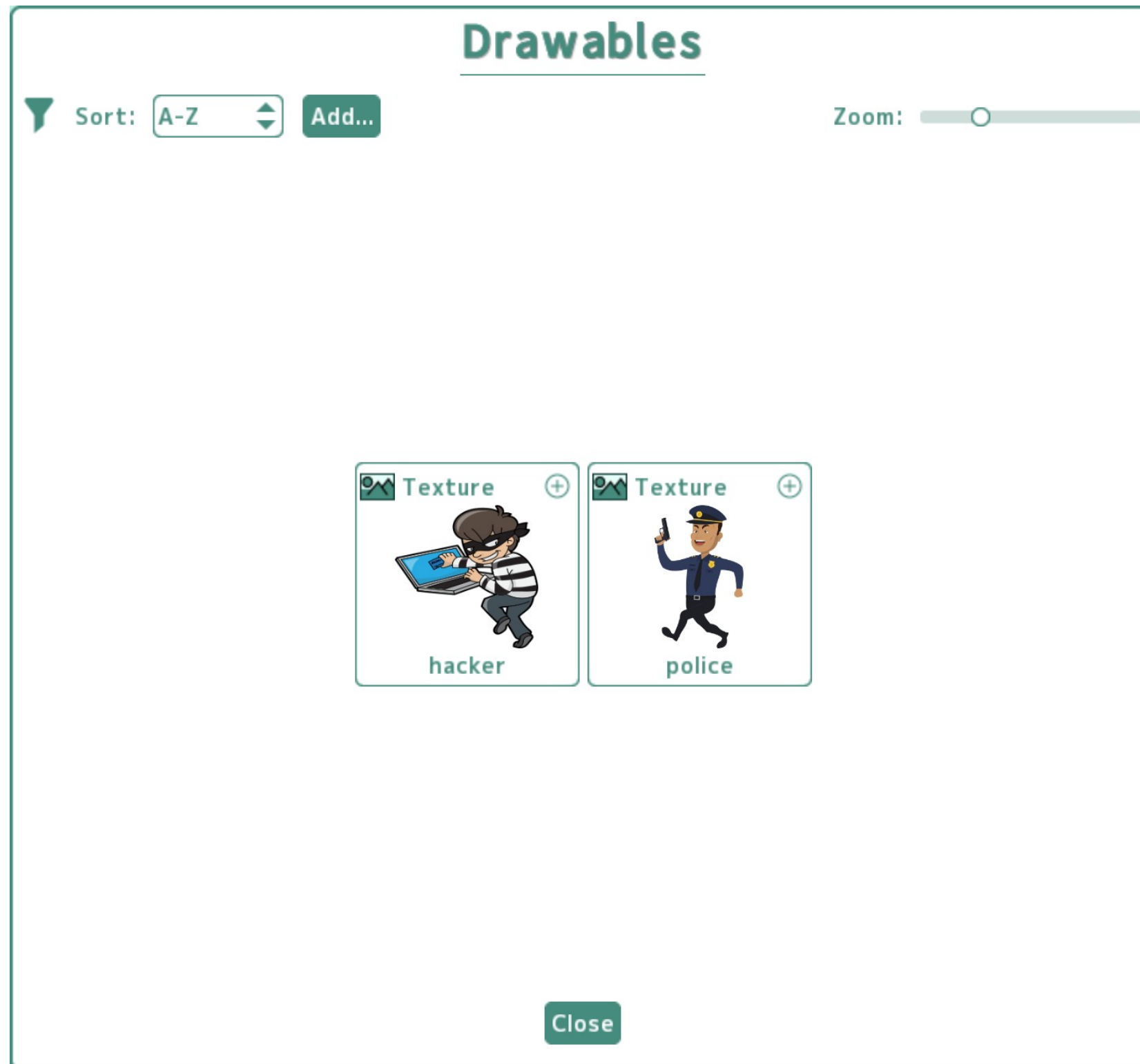
- The ability of OpenGL to render fragments of a texture allows multiple images to be packed together in a (large) single texture, which in LibGDX is called TextureAtlas. These fragments (TextureRegions) can be drawn independently in the same batch draw.
- If images of individual alphanumeric characters are packed together in a Texture Atlas, they can be used to render text in games. This is done using BitmapFont class.
- Animation frames can be packed together in a TextureAtlas and displayed at the appropriate frame rate during the game. This is done using Animation class
- A drawable object such as TextureRegion can be extended with attributes such as its location in the world, orientation, and size, which are used when displaying the image. This is the idea behind Sprite class.

TextureAtlas

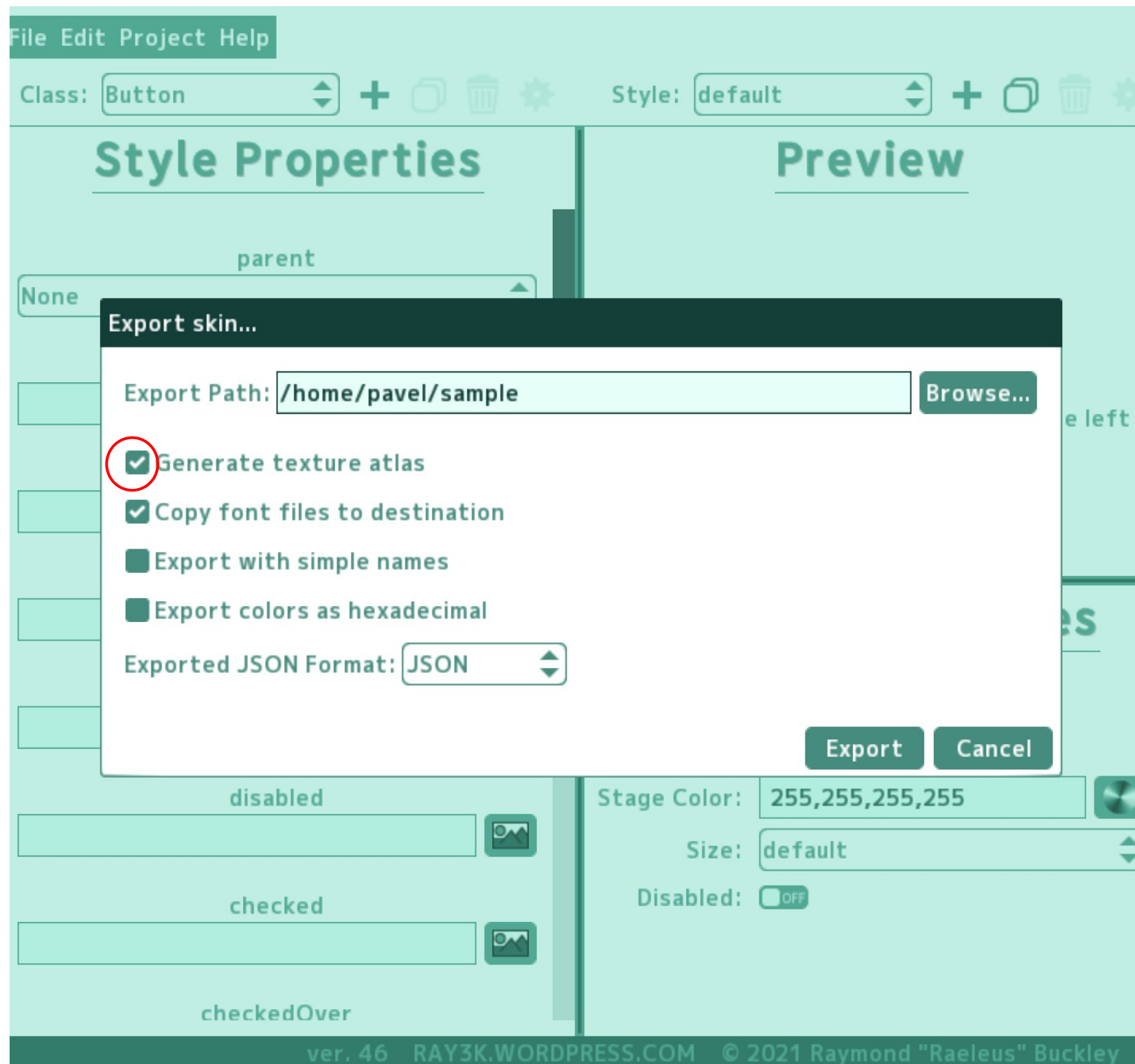
A texture atlas is described using two files:

- An image file (typically a .png file)
- A text file (typically has file extension .atlas) describing the name, index, and coordinates of individual pictures within the image file. In some cases, the images could also be rotated from their original orientation.
- Texture atlases can be produced using SkinComposer:
 - 1) Create New empty skin project
 - 2) Add pictures in the Drawables section of the SkinComposer project
 - 3) Export data with the option to write .atlas files

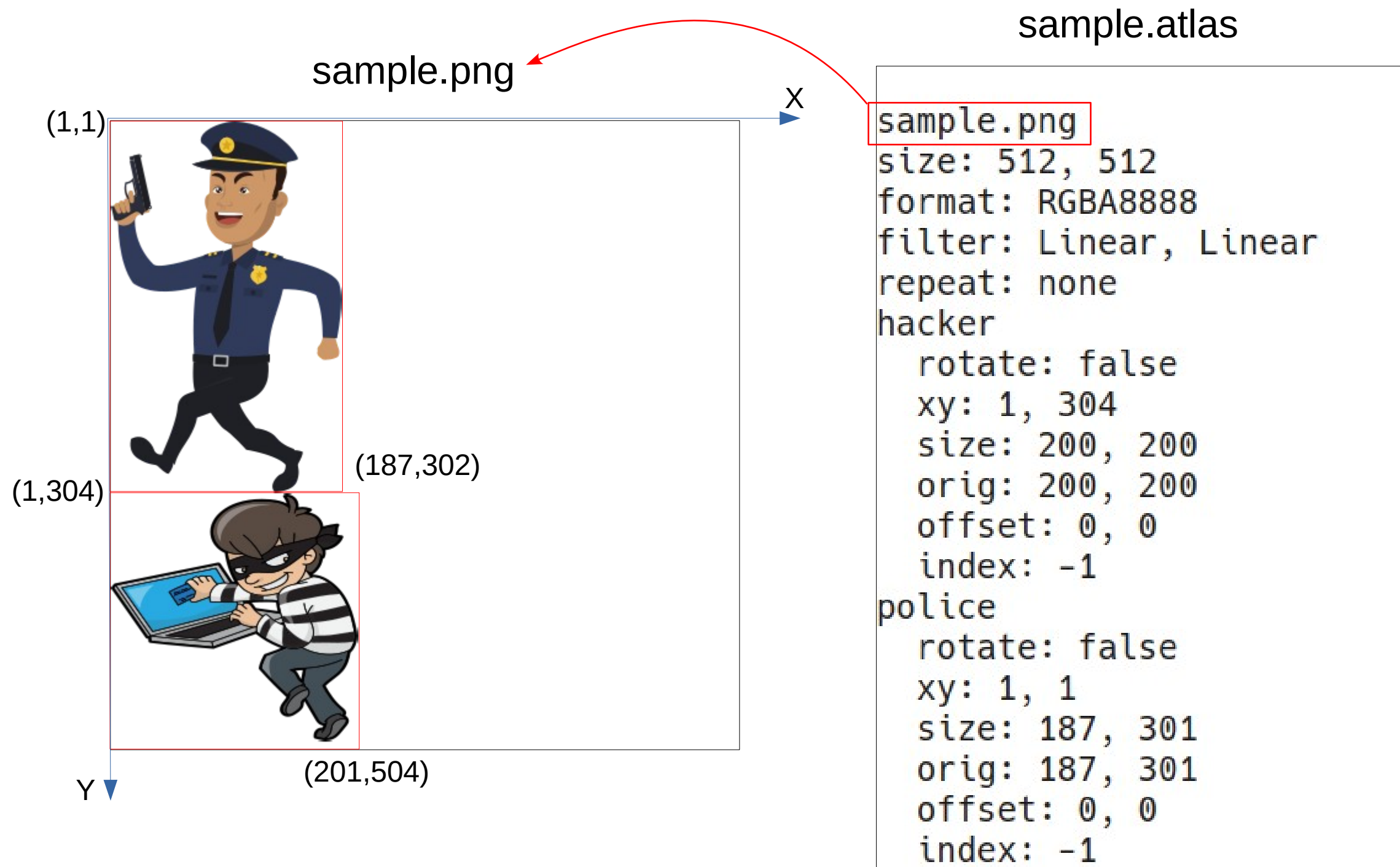
Drawables window in SkinComposer



Exporting game resources from SkinComposer



Texture Atlas files



TextureAtlas class usage

```
TextureAtlas atlas;  
TextureAtlas.AtlasRegion hacker;  
TextureAtlas.AtlasRegion police;
```

} Class fields

```
atlas = new TextureAtlas(Gdx.files.internal("sample.atlas"));  
hacker = atlas.findRegion("hacker");  
police = atlas.findRegion("police");
```

} in create()

```
batch.begin();  
batch.draw(hacker,0,0);  
batch.draw(hacker,10,10);  
batch.draw(hacker,20,20);  
batch.draw(police,300,0);  
batch.end();
```

} in render()

```
atlas.dispose();
```

} in dispose()

TextureAtlas usage example: complete code

```
public class MyGame extends ApplicationAdapter {
    TextureAtlas atlas;
    TextureAtlas.AtlasRegion hacker;
    TextureAtlas.AtlasRegion police;
    SpriteBatch batch;
    OrthographicCamera camera;

    @Override
    public void create() {
        atlas = new TextureAtlas(Gdx.files.internal("sample.atlas"));
        hacker = atlas.findRegion("hacker");
        police = atlas.findRegion("police");
        batch = new SpriteBatch();
        camera = new OrthographicCamera();
    }

    @Override
    public void render() {
        ScreenUtils.clear(1.0f, 1.0f, 1.0f, 1.0f);
        camera.setToOrtho(false, Gdx.graphics.getWidth(), Gdx.graphics.getHeight());
        camera.update();
        batch.setProjectionMatrix(camera.combined);
        batch.begin();
        batch.draw(hacker, 0, 0);
        batch.draw(hacker, 10, 10);
        batch.draw(hacker, 20, 20);
        batch.draw(police, 300, 0);
        batch.end();
    }

    @Override
    public void dispose() {
        batch.dispose();
        atlas.dispose();
    }
}
```

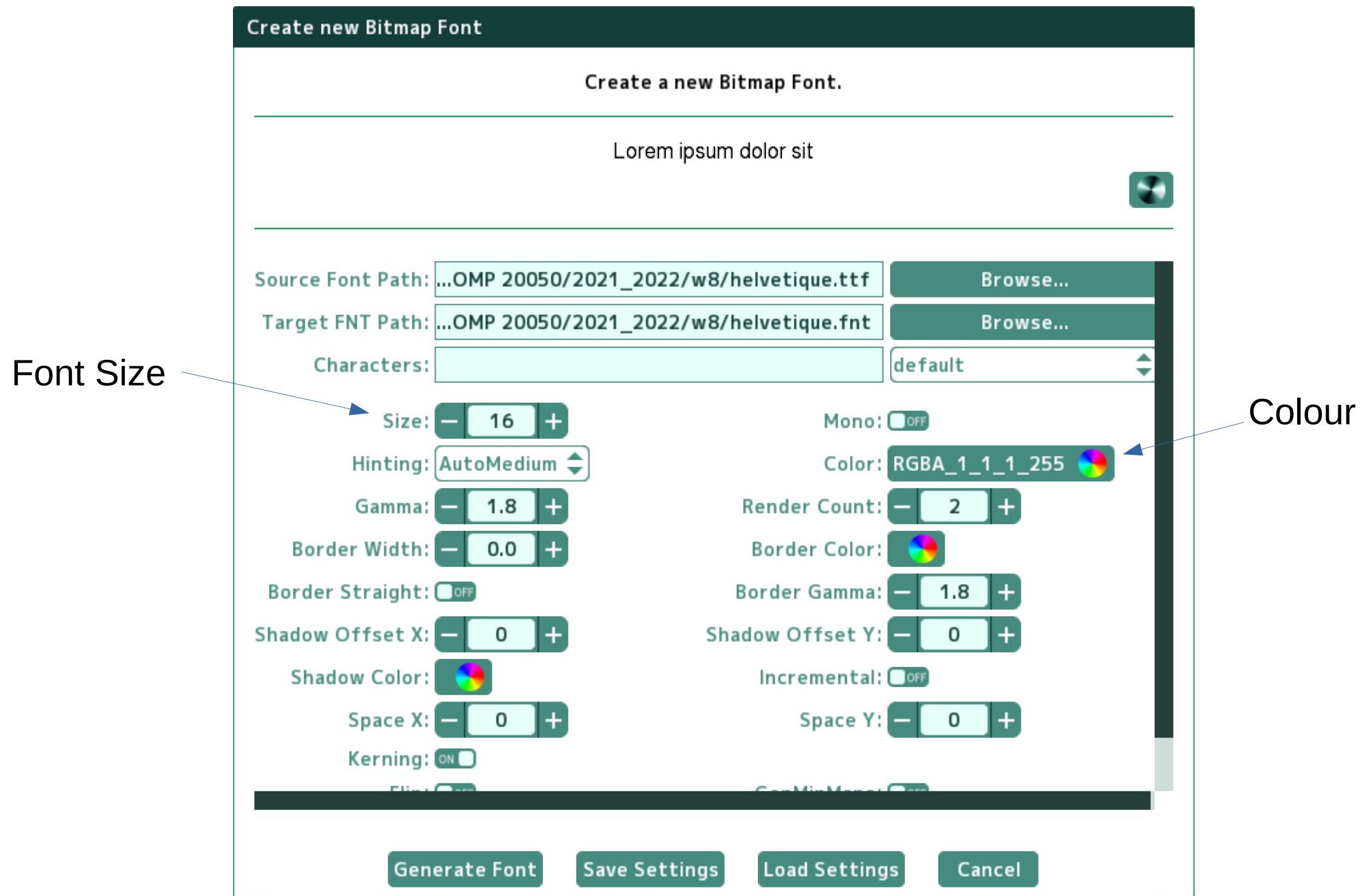
TextureAtlas usage example output



BitmapFont

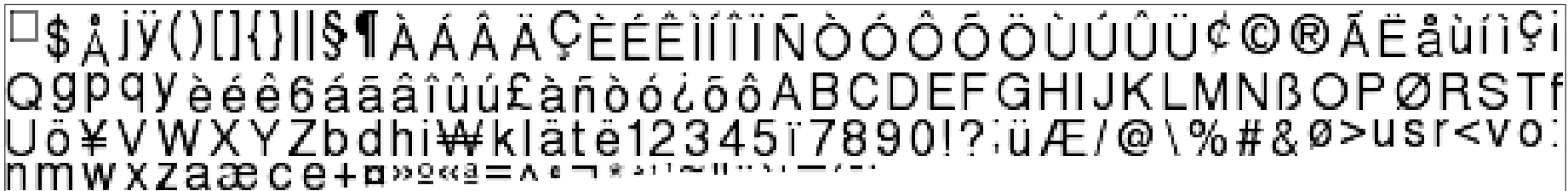
- A specialised texture atlas that contains pictures of letteres and digits
- BitmapFonts can be generated from True Type Fonts using SkinComposer
- BitmapFonts can be included into Skins along with textures and UI elements.
- Bitmap fonts need to be disposed of in the dispose() method of the game.

Font creation window in SkinComposer



Bitmap font files

helvetique.png



helvetique.fnt

```
[info face="helvetique" size=12 bold=0 italic=0 charset=""
unicode=1 stretchH=100 smooth=1 aa=2 padding=1,1,1,1 spacing=0,0
common lineHeight=16 base=11 scaleW=506 scaleH=61 pages=1
packed=0 alphaChnl=0 redChnl=0 greenChnl=0 blueChnl=0
page id=0 file="helvetique.png"
chars count=181
char id=0      x=2      y=2      width=9      height=10     xoffset=1
yoffset=1      xadvance=12    page=0      chnl=0
char id=32     x=0      y=0      width=0      height=0      xoffset=-1
yoffset=-1     xadvance=4     page=0      chnl=0
char id=33     x=304    y=37     width=3      height=12     xoffset=0
yoffset=-1     xadvance=4     page=0      chnl=0
char id=34     x=229    y=51     width=6      height=4      xoffset=-1
yoffset=-1     xadvance=6     page=0      chnl=0
...

```


BitmapFont class usage

BitmapFont **helvetique**; } Class field

helvetique = new BitmapFont(Gdx.*files*.internal(**"helvetique.fnt"**));
in create()

batch.begin();
helvetique.draw(**batch**, **"Hello World!"**, **10**, **200**);
batch.end();

X Y

} in render()

helvetique.dispose(); } in dispose()

BitmapFont usage example: complete code

```
public class MyGame extends ApplicationAdapter {
    BitmapFont helvetique;
    SpriteBatch batch;
    OrthographicCamera camera;

    @Override
    public void create() {
        helvetique = new BitmapFont(Gdx.files.internal("helvetique.fnt"));
        batch = new SpriteBatch();
        camera = new OrthographicCamera();
    }

    @Override
    public void render() {
        ScreenUtils.clear(1.0f, 1.0f, 1.0f, 1.0f);
        camera.setToOrtho(false, Gdx.graphics.getWidth(), Gdx.graphics.getHeight());
        camera.update();
        batch.setProjectionMatrix(camera.combined);
        batch.begin();
        helvetique.draw(batch, "Hello World!", 10, 200);
        batch.end();
    }

    @Override
    public void dispose() {
        batch.dispose();
        helvetique.dispose();
    }
}
```

BitmapFont usage example: complete code


```
public class MyGame extends ApplicationAdapter {
    BitmapFont helvetique;
    SpriteBatch batch;
    OrthographicCamera camera;

    @Override
    public void create() {
        helvetique = new BitmapFont(Gdx.files.internal("helvetique.fnt"));
        batch = new SpriteBatch();
        camera = new OrthographicCamera();
    }

    @Override
    public void render() {
        ScreenUtils.clear(1.0f, 1.0f, 1.0f, 1.0f);
        camera.setToOrtho(false, Gdx.graphics.getWidth(), Gdx.graphics.getHeight());
        camera.update();
        batch.setProjectionMatrix(camera.combined);
        batch.begin();
        helvetique.draw(batch, "Hello World!", 10, 200);
        batch.end();
    }

    @Override
    public void dispose() {
        batch.dispose();
        helvetique.dispose();
    }
}
```

BitmapFont usage example



Hello World!

Animation class

Think of it as a stand-alone picture, whose display changes over time.

When an animation object is created, it is given

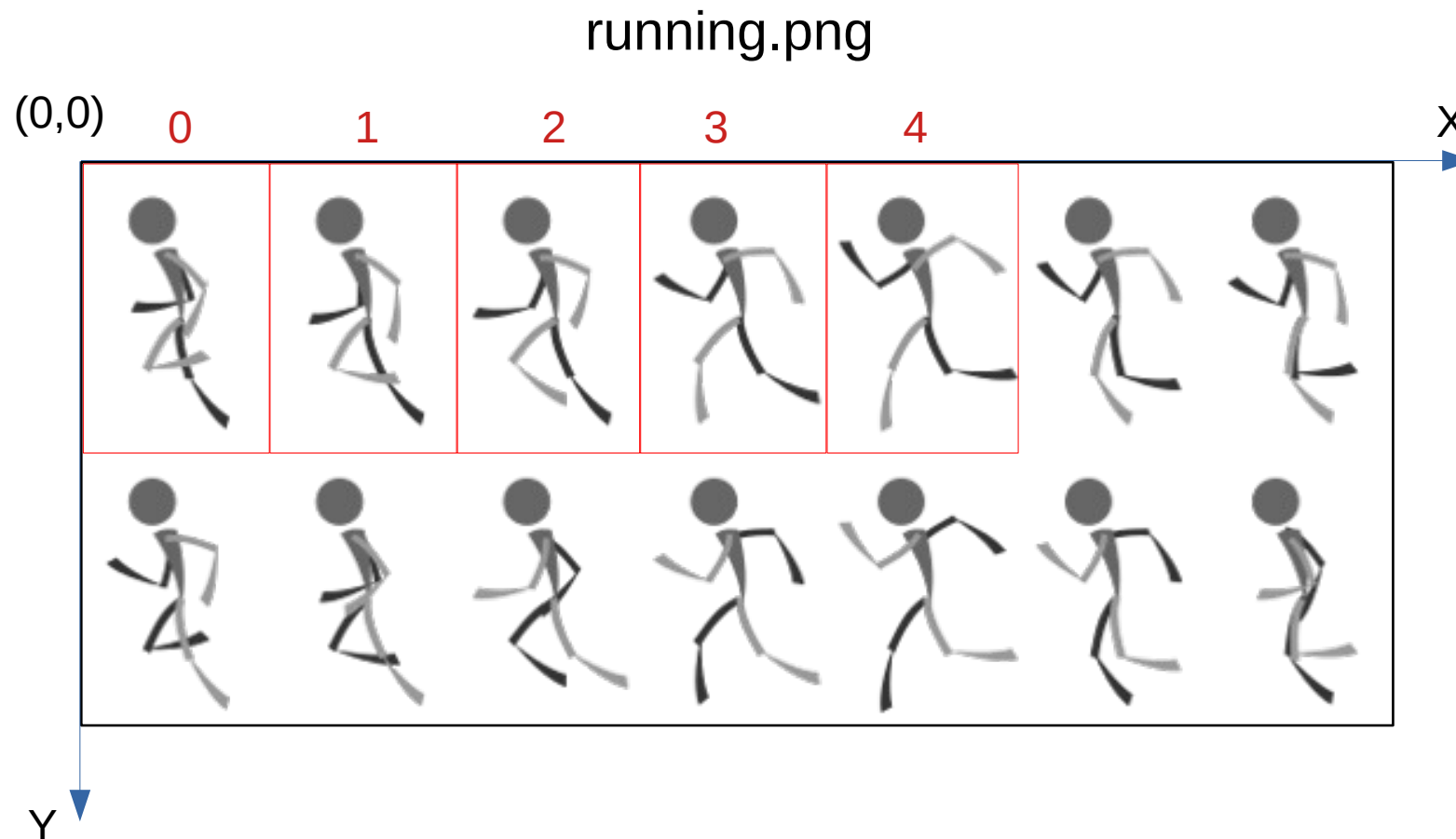
- A frame rate that specifies how often frames replace each other.
- An array of texture regions, where each texture region corresponds to an animation frame
- Animation play mode: e.g. play it once and stop, loop the animation, etc.

In the `render()` method, the programmer must keep time elapsed since the start of animation. For example, we could create a field called *elapsedTime*, which is set to 0 at the start of animation and incremented by `Gdx.graphics.getDeltaTime()` in each `render()` call.

To get the appropriate animation frame in a particular `render()` call, we use Animation class method `getKeyFrame`:

TextureAtlas.AtlasRegion `getKeyFrame(float elapsedTime);`

Indexed texture atlas for Animation



All animation frames have the same name, but each animation frame has unique index, which represents its sequence number in the animation (0,1,2,3,4...)

Note that the .atlas file is just a text file and its contents can be modified with a text editor.

running.atlas

```
running.png
format: RGBA8888
filter: Linear,Linear
repeat: none
running
  rotate: false
  xy: 0, 0
  size: 80, 120
  orig: 80, 120
  offset: 0, 0
  index: 0
running
  rotate: false
  xy: 80, 0
  size: 80, 120
  orig: 80, 120
  offset: 0, 0
  index: 1
running
  rotate: false
  xy: 160, 0
  size: 80, 120
  orig: 80, 120
  offset: 0, 0
  index: 2
running
  rotate: false
  xy: 240, 0
  size: 80, 120
  orig: 80, 120
  offset: 0, 0
  index: 3
running
  rotate: false
  xy: 320, 0
  size: 80, 120
  orig: 80, 120
  offset: 0, 0
  index: 4
...
```

Animation class usage

```
static final float FRAME_DURATION = 1/10.0f;  
TextureAtlas runningAtlas;  
Animation runningAnimation;  
float elapsedTime;
```

} Class fields

```
runningAtlas = new TextureAtlas(Gdx.files.internal("running.atlas"));  
Array<TextureAtlas.AtlasRegion> runningFrames = runningAtlas.findRegions("running");  
runningAnimation = new Animation(FRAME_DURATION, runningFrames, Animation.PlayMode.LOOP);  
elapsedTime = 0;
```

} in create()

```
elapsedTime = elapsedTime + Gdx.graphics.getDeltaTime();  
TextureRegion currentFrame = (TextureRegion) runningAnimation.getKeyFrame(elapsedTime);  
...  
batch.draw(currentFrame, 300, 40);
```

} in render()

X Y

```
runningAtlas.dispose(); } in dispose()
```

Animaiton example: complete code

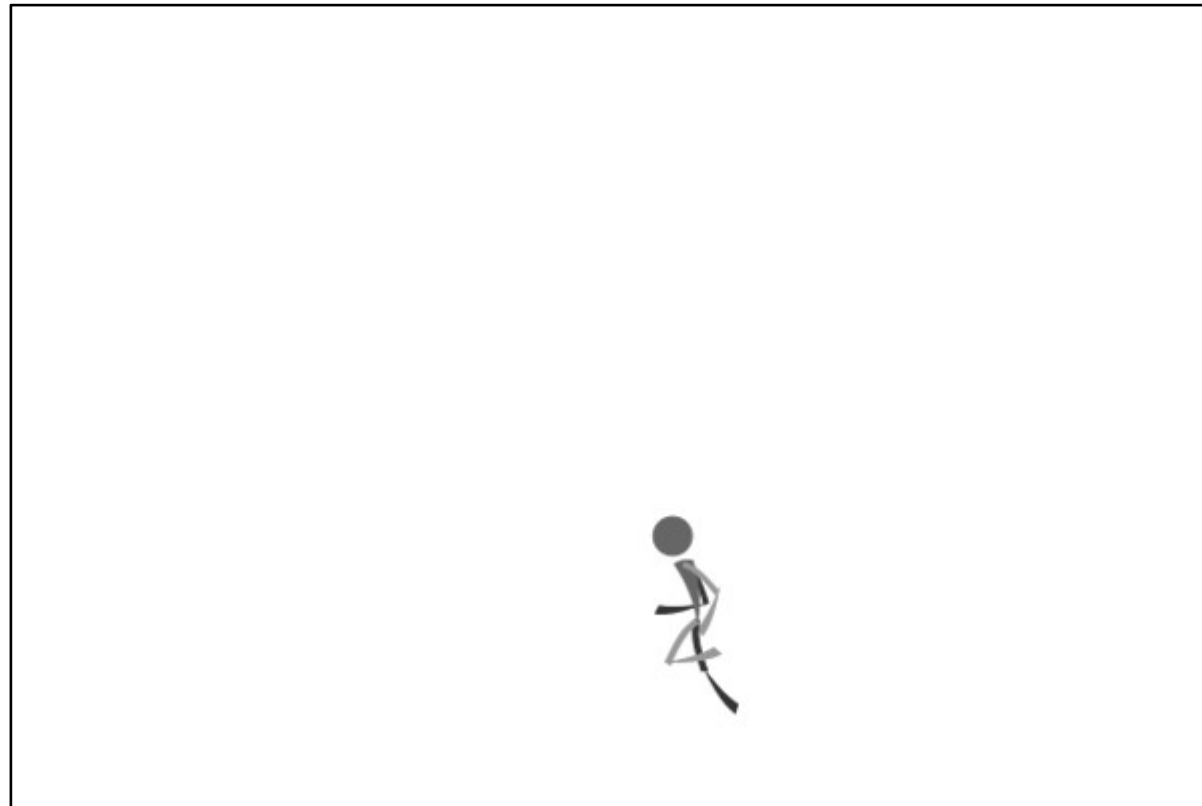
```
public class MyGame extends ApplicationAdapter {
    static final float FRAME_DURATION = 1/10.0f; // 10 frames per second
    TextureAtlas runningAtlas;
    Animation runningAnimation;
    float runnerX, runnerY, elapsedTime;
    SpriteBatch batch;
    OrthographicCamera camera;

    @Override
    public void create() {
        runningAtlas = new TextureAtlas(Gdx.files.internal("running.atlas"));
        Array<TextureAtlas.AtlasRegion> runningFrames = runningAtlas.findRegions("running");
        runningAnimation = new Animation(FRAME_DURATION, runningFrames, Animation.PlayMode.LOOP);
        runnerX = Gdx.graphics.getWidth() * 0.5f;
        runnerY = Gdx.graphics.getHeight() * 0.1f;
        elapsedTime = 0;
        batch = new SpriteBatch();
        camera = new OrthographicCamera();
    }

    @Override
    public void render() {
        ScreenUtils.clear(1.0f, 1.0f, 1.0f, 1.0f);
        elapsedTime = elapsedTime + Gdx.graphics.getDeltaTime();
        TextureRegion currentFrame = (TextureRegion) runningAnimation.getKeyFrame(elapsedTime);
        camera.setToOrtho(false, Gdx.graphics.getWidth(), Gdx.graphics.getHeight());
        camera.update();
        batch.setProjectionMatrix(camera.combined);
        batch.begin();
        batch.draw(currentFrame, runnerX, runnerY);
        batch.end();
    }

    @Override
    public void dispose() {
        batch.dispose();
        runningAtlas.dispose();
    }
}
```


Animation example



Sprite class

Sprite class encapsulates texture region together with its position on the world plane, rotation angle, colour, etc. (see Sprite class details in javadocs section of <https://libgdx.com/dev/>).

A sprite object can `translate()` along X and Y axes, `rotate()`, `scale()`, change its colour with `setColor()` among other things.

Proprietary Sprite-like classes can be created to hold individual gamepieces.

Sprite class usage example

Sprite **officer**; } Class field

```
atlas = new TextureAtlas(Gdx.files.internal("sample.atlas"));  
police = atlas.findRegion("police");  
officer = new Sprite(police);  
officer.setPosition(START_X, START_Y);
```

} in create()

```
officer.translateX(SPEED * Gdx.graphics.getDeltaTime());  
...  
officer.draw(batch);
```

} in render()

```
atlas.dispose();
```

} in dispose()

Sprite example: complete code

```
public class MyGame extends ApplicationAdapter {
    final static float SPEED = -80.0f; //moving 80 world units per second to the left
    final static float START_X = 400.0f; // starting X coordinate
    final static float START_Y = 10.0f; // starting Y coordinate

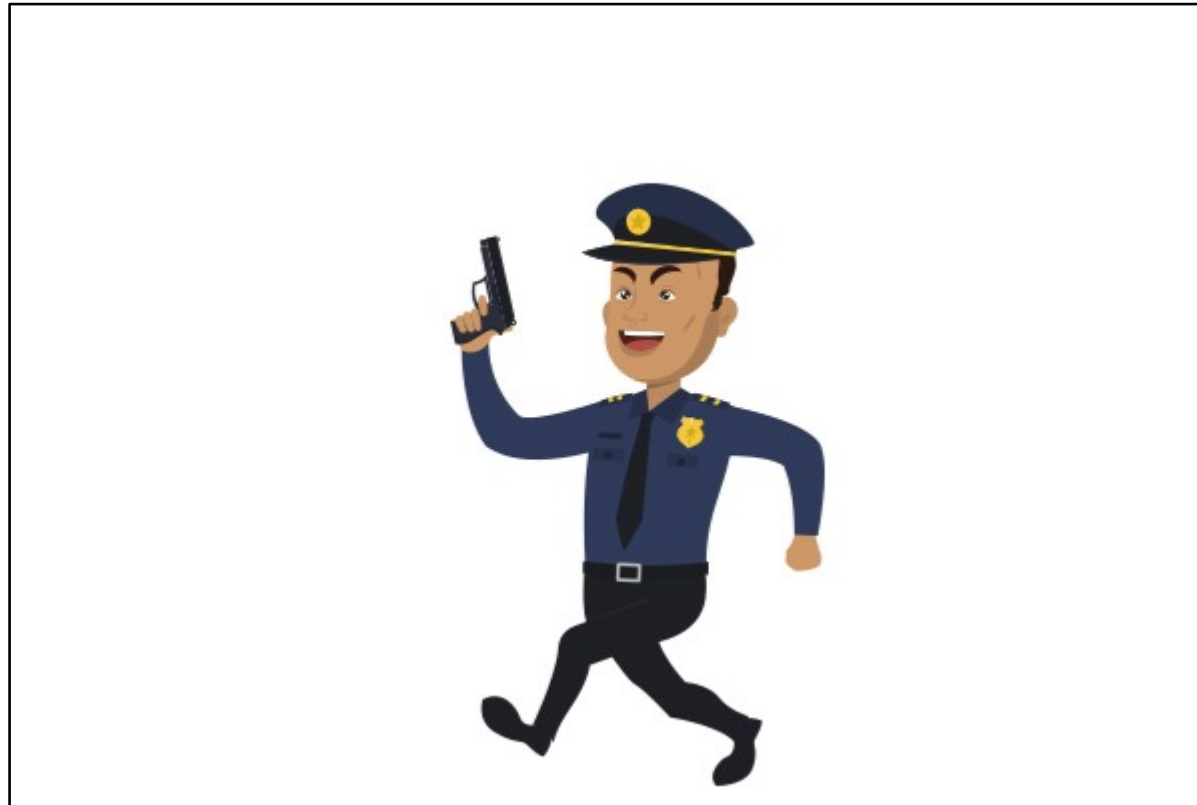
    TextureAtlas atlas;
    TextureAtlas.AtlasRegion police;
    Sprite officer;
    SpriteBatch batch;
    OrthographicCamera camera;

    @Override
    public void create() {
        atlas = new TextureAtlas(Gdx.files.internal("sample.atlas"));
        police = atlas.findRegion("police");
        officer = new Sprite(police);
        officer.setPosition(START_X, START_Y);
        batch = new SpriteBatch();
        camera = new OrthographicCamera();
    }

    @Override
    public void render() {
        ScreenUtils.clear(1.0f, 1.0f, 1.0f, 1.0f);
        camera.setToOrtho(false, Gdx.graphics.getWidth(), Gdx.graphics.getHeight());
        camera.update();
        officer.translateX(SPEED * Gdx.graphics.getDeltaTime());
        if (officer.getX() < 0) {
            officer.setX(START_X);
        }
        batch.setProjectionMatrix(camera.combined);
        batch.begin();
        officer.draw(batch);
        batch.end();
    }

    @Override
    public void dispose() {
        batch.dispose();
        atlas.dispose();
    }
}
```

Sprite example



Skin class

Although Skin class is part of the more advanced Scene2D package, it provides convenient way to package multiple images, fonts, colours, and GUI widget elements as a single large texture.

Skins can be created with SkinComposer and exported as .json files, which references its constituent .png .fnt and .atlas files.

Skin class has many methods to instantiate various types of resources that we explored so far, for example:

```
TextureRegion  getRegion(String name)
```

```
BitmapFont    getFont(String name)
```

```
Array<TextureRegion>  getRegions(String name)
```

We will see an example of using skin to store resources towards the end of this chapter.

Some points regarding running SkinComposer.jar

- LibGDX documentation page: <https://libgdx.com/wiki/tools/skin-composer>
- Download link: <https://github.com/raeleus/skin-composer/releases>
- Does not work with Java 8, requires a more recent version (I use JDK 17)
- Start from command line: **java -jar SkinComposer.jar**
- In addition to the three default Skin project immediately available when you start SkinComposer, a large number of additional pre-made skins are available at <https://ray3k.wordpress.com/artwork/>
 - Download .ZIP of the skin and unzip it somewhere.
 - In SkinComposer go to Files -> Import ...
 - You can add custom images, fonts, etc.
 - Save your skin project somewhere.
 - Export Skin files for your game project (skin PNG file, skin JSON file, .atlas files, .fnt files) as usual. These exported files is all that needs to be in the resources/ folder of your game.

LibGDX Input

LibGDX input

Different platforms have different input facilities.

On the desktop users can talk to your application via the keyboard and a mouse. The same is true for browser based games.

On Android and iOS, the mouse is replaced with a (capacitive) touch screen, and a hardware keyboard is often missing. All (compatible) Android devices also feature an accelerometer and sometimes even a compass (magnetic field sensor).

libGDX abstract all these different input devices. **Mouse and touch screens are treated as being the same, with mice lacking multi-touch support (they will only report a single “finger”) and touch-screens lacking button support (they will only report “left button” presses).**

Depending on the input device, one can either poll the state of a device periodically, or register a listener that will receive input events in chronological order.

(source: <https://libgdx.com/wiki/input/input-handling>)

LibGDX polled input

The simplest way to get user input is to poll the current state. Here are some handy functions:

- **Gdx.input.isTouched()** returns true or false depending on whether the user is currently clicking the mouse or touching the screen. This works across devices on both touch screens and mouse interfaces.
- **Gdx.input.getX()** returns the X value of the touch or click.
- **Gdx.input.getY()** returns the Y value of the touch or click.
- **Gdx.input.isKeyPressed()** returns whether a particular key is currently pressed. This function takes an int key argument, which you can find in the Input.Keys class. For example **Gdx.input.isKeyPressed(Input.Keys.W)** checks whether the W key is currently pressed. This function only makes sense on devices that have a physical keyboard.

(source: <https://happycoding.io/tutorials/libgdx/input>)

Polled input example (see polled-input-example.zip)

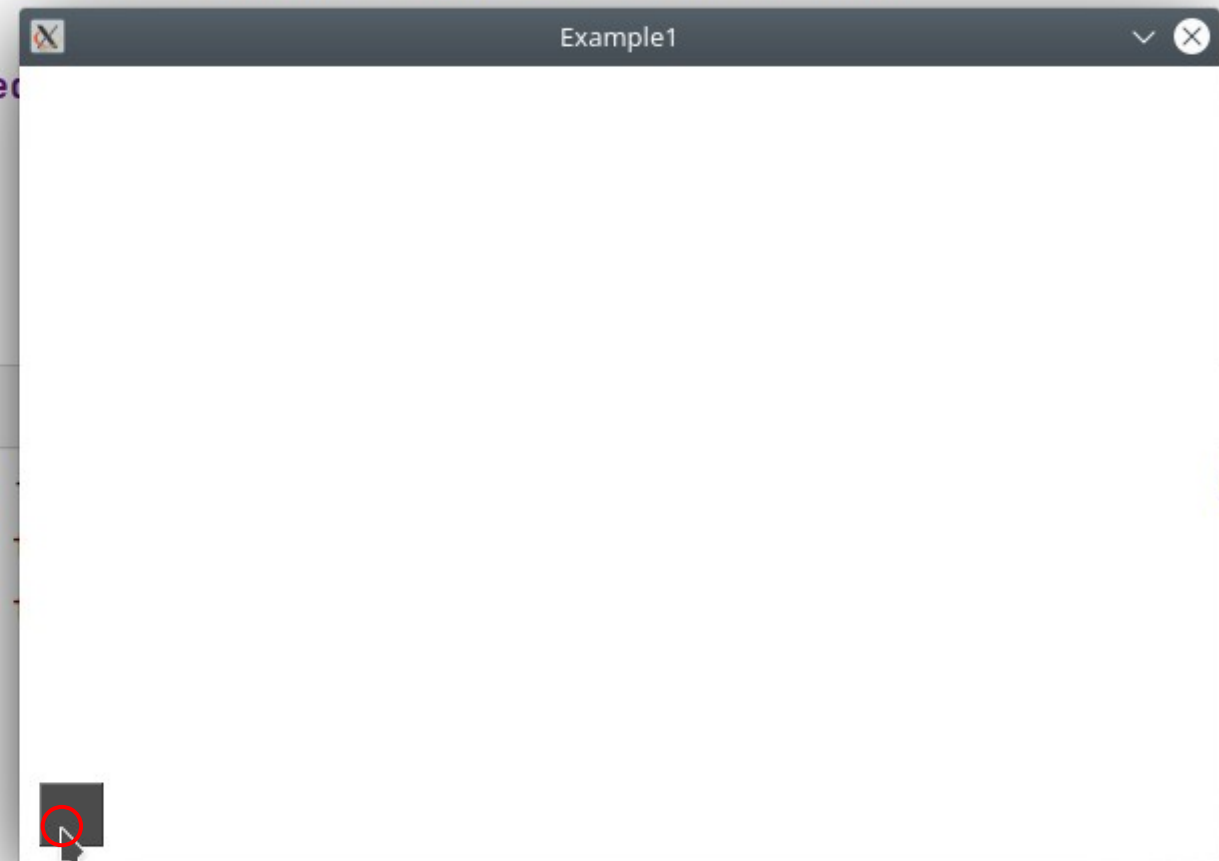
```
public void render() {  
    if (Gdx.input.isTouched()) {  
        System.out.printf("x=%d, y=%d\n", Gdx.input.getX(), Gdx.input.getY());  
    }  
    ScreenUtils.clear(1.0f, 1.0f, 1.0f, 1.0f);  
    camera.setToOrtho(false, Gdx.graphics.getWidth(), Gdx.graphics.getHeight());  
    camera.update();  
    batch.setProjectionMatrix(camera.combined);  
    batch.begin();  
    batch.draw(square, 10, 10);  
    batch.end();  
}
```

Polled input example output

```
21  @Override
22  public void render() {
23      if (Gdx.input.isTouched()) {
24          System.out.printf(s: "x=%d, y=%d\n", Gdx.input.getX(), Gdx.input.getY());
25      }
26      ScreenUtils.clear(r: 1.0f, g: 1.0f, b: 1.0f, a: 1.0f);
27      camera.setToOrtho(yDown: false, Gdx.graphics.getWidth(), Gdx.graphics.getHeight());
28      camera.update();
29      batch.setProjectionMatrix(camera.combined);
30      batch.begin();
31      batch.draw(square, x: 10, y: 10);
32      batch.end();
33  }
```

Run: batch-drawing-example-1.0-SNAPSHOT-all.jar

```
/usr/lib/jvm/java-1.8.0-openjdk-amd64/bin/java
[ALSOFT] (EE) Failed to set real-time priority
[ALSOFT] (EE) Failed to set real-time priority
x=19, y=380
x=19, y=380
x=19, y=380
x=19, y=380
x=19, y=380
x=19, y=380
x=19, y=380
x=19, y=380
x=19, y=380
```



Version Control Run TODO Problems Terminal Build Dependencies

Gradle sync finished in 6 s 724 ms (18 minutes ago)

Input handling - libGDX — Mozilla ... batch-drawing-example - My...

comp20050_a3 — Dolphin

Example1

Why input X, Y coordinates are different from world coordinates?

Gdx input always returns mouse pointer coordinates in Screen coordinates, but the Square is drawn on the world plane using world coordinates, and its image is then projected onto the screen:

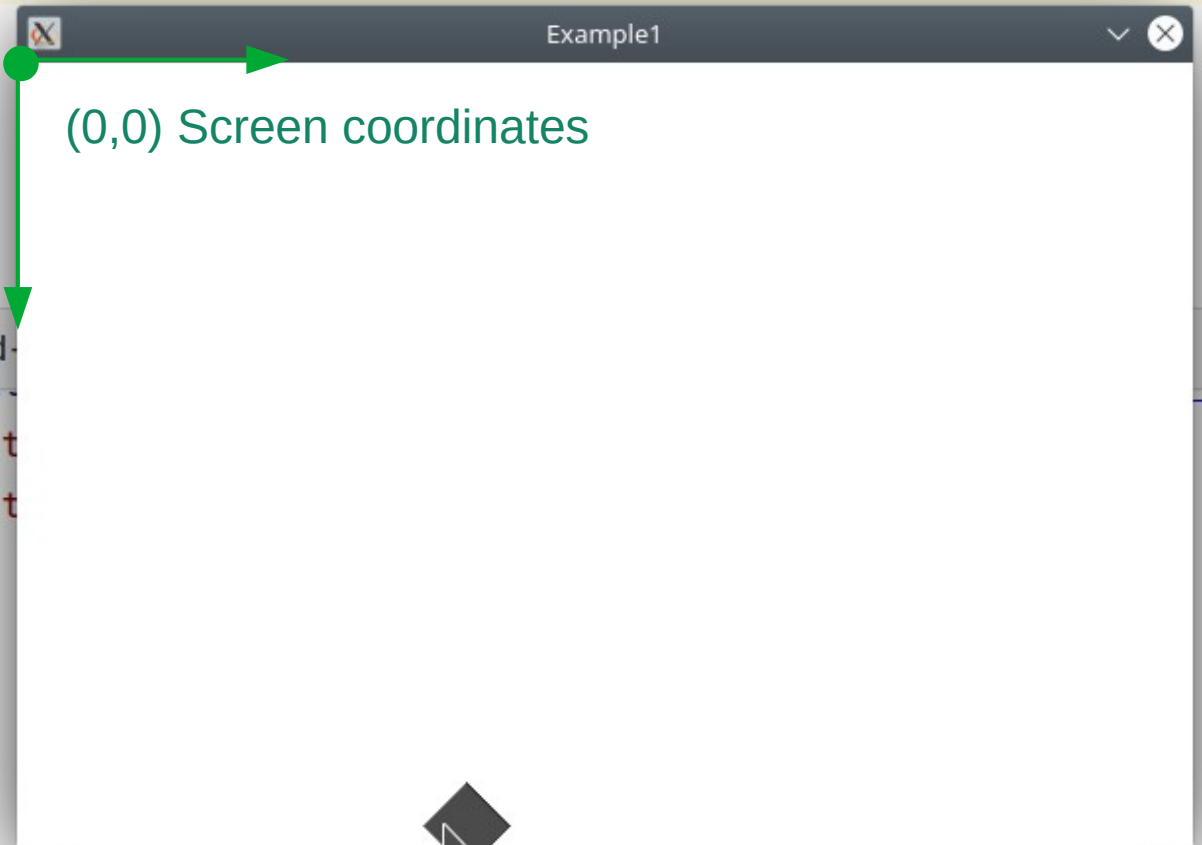


Things get even more complicated if we manipulate the camera...

```
21  @Override
22  public void render() {
23      if (Gdx.input.isTouched()) {
24          System.out.printf(s: "x=%d, y=%d\n", Gdx.input.getX(), Gdx.input.getY());
25      }
26      ScreenUtils.clear(r: 1.0f, g: 1.0f, b: 1.0f, a: 1.0f);
27      camera.setToOrtho(yDown: false, Gdx.graphics.getWidth(), Gdx.graphics.getHeight());
28      camera.rotate(angle: 45.0f);
29      camera.update();
30      batch.setProjectionMatrix(camera.combined);
31      batch.begin();
32      batch.draw(square, x: 100, y: 100);
33      batch.end();
```

Run: batch-drawing-example-1.0-SNAPSHOT-all.jar x polled-

[ALSOFT] (EE) Failed to set real-time priority for t
[ALSOFT] (EE) Failed to set real-time priority for t
x=218, y=392
x=218, y=392
x=218, y=392
x=218, y=392
x=218, y=392
x=218, y=392
x=218, y=392
x=218, y=392



Version Control Run TODO Problems Terminal Build Dependencies

Gradle sync finished in 6 s 724 ms (35 minutes ago) World coordinates (0,0)

“Unprojecting” screen coordinates into the world

We need a way to relate screen coordinates back to the world coordinates, in order to determine which game object was pointed at by the user.

How can we do it? - we can use camera's combined matrix to “unproject” a point in screen coordinates back to the world plane!

1. Create a Vector3 object that holds screen coordinates of the point (since screen is 2D, we set z coordinate to 0):

```
Vector3 screen = new Vector3(Gdx.input.getX(), Gdx.input.getY(),0);
```

2. Use camera's unproject method to get a corresponding point in the world coordinates:

```
Vector3 world = camera.unproject(screen);
```

world.x and world.y are the coordinates of the screen point on the world plane.

Result of unprojection (Note that the world coordinates are fractional number of the type float !)

```
29 batch.begin();
30 batch.draw(square, x: 10, y: 10);
31 batch.end();
32 if (Gdx.input.isTouched()) {
33     Vector3 screen = new Vector3(Gdx.input.getX(), Gdx.input.getY(), z: 0);
34     Vector3 world = camera.unproject(screen);
35     System.out.printf(s: "x=%f, y=%f\n", world.x, world.y);
36 }
37 }
```

38 @Override

Run: polled-input-example [shadow]

[ALSOFT] (EE) Failed to s
[ALSOFT] (EE) Failed to s
x=21.000000, y=19.000000
x=21.000000, y=19.000000
x=21.000000, y=19.000000
x=21.000000, y=19.000000
x=21.000000, y=19.000000
x=21.000000, y=19.000000
x=21.000000, y=19.000000
x=21.000000, y=19.000000
x=21.000000, y=19.000000
x=21.000000, y=19.000000

Example1

(0,0) Screen coordinates

(0,0)
World coordinates

The image shows a Java IDE with a code editor and a console window. The code editor displays a snippet of Java code that uses Gdx for input handling and unprojection. The console window shows the output of the code, which is a series of floating-point numbers representing world coordinates. A diagram is overlaid on the console window, showing a green arrow pointing from the top-left corner of the console window to the text "(0,0) Screen coordinates". A red arrow points from the bottom-left corner of the console window to the text "(0,0) World coordinates".

Input events

Polling for events will work for things like continuous input (tracking the mouse or touch position), but can be cumbersome for responding to user input (clicking, tapping, or typing a key). For these cases, you can use an input event handler.

To create an input event handler, either implement **InputProcessor** (if you care about every event) or extend **InputAdapter** (if you only care about some events) and then call the **Gdx.input.setInputProcessor()** function with your event handler.

(source: <https://happycoding.io/tutorials/libgdx/input>)

InputProcessor interface

```
public interface InputProcessor {  
  
    public boolean keyDown (int keycode);  
    public boolean keyUp (int keycode);  
    public boolean keyTyped (char character);  
    public boolean touchDown (int screenX, int screenY, int pointer, int button);  
    public boolean touchUp (int screenX, int screenY, int pointer, int button);  
    public boolean touchDragged (int screenX, int screenY, int pointer);  
    public boolean mouseMoved (int screenX, int screenY);  
    public boolean scrolled (float amountX, float amountY);  
  
}
```

MyEventProcessor

```
public class MyEventHandler extends InputAdapter {  
  
    @Override  
    public boolean touchDown(int screenX, int screenY, int pointer, int button) {  
        System.out.printf("touchDown event: x=%d, y=%d, pointer=%d, button=%d\n",  
                           screenX, screenY, pointer, button);  
        return true;  
    }  
}
```

MyGame with MyInputProcessor

```
public class MyGame extends ApplicationAdapter {

    Texture square;
    SpriteBatch batch;
    OrthographicCamera camera;

    @Override
    public void create() {
        square = new Texture(Gdx.files.internal("game_square_black.png"));
        batch = new SpriteBatch();
        camera = new OrthographicCamera();
        Gdx.input.setInputProcessor(new MyEventHandler());
    }

    @Override
    public void render() {
        ScreenUtils.clear(1.0f, 1.0f, 1.0f, 1.0f);
        camera.setToOrtho(false, Gdx.graphics.getWidth(), Gdx.graphics.getHeight());
        camera.update();
        batch.setProjectionMatrix(camera.combined);
        batch.begin();
        batch.draw(square, 10, 10);
        batch.end();
        if (Gdx.input.isTouched()) {
            Vector3 screen = new Vector3(Gdx.input.getX(), Gdx.input.getY(), 0);
            Vector3 world = camera.unproject(screen);
            System.out.printf("x=%f, y=%f\n", world.x, world.y);
        }
    }

    @Override
    public void dispose() {
        batch.dispose();
        square.dispose();
    }
}
```

Input Events example output

The screenshot shows an IDE with a code editor and a console window. The code editor displays a Java class with a `render()` method that checks for touch events and prints their coordinates. The console window shows the output of the program, which is a series of `touchDown` events at the same coordinates (x=20, y=384). A red box highlights the first event, and a red arrow points to it with the text "Only one touchDown event per mouse click!".

```
23 @Override
24 public void render() {
25     ScreenUtils.clear(r: 1.0f, g: 1.0f, b: 1.0f, a: 1.0f);
26     camera.setToOrtho(yDown: false, Gdx.graphics.getWidth(), Gdx.graphics.getHeight());
27     camera.update();
28     batch.setProjectionMatrix(camera.combined);
29     batch.begin();
30     batch.draw(square, x: 10, y: 10);
31     batch.end();
32     if (Gdx.input.isTouched()) {
33         System.out.printf(s: "x=%d, y=%d\n", Gdx.input.getX(), Gdx.input.getY());
34     }
35 }
36
```

Run: polled-input-example-1.0-SNAPSHOT-all.jar x input-ever

touchDown event: x=20, y=384, pointer=0, button=0

x=20, y=384

x=20, y=384

x=20, y=384

x=20, y=384

x=20, y=384

x=20, y=384

x=20, y=384

x=20, y=384

x=20, y=384

x=20, y=384

Only one touchDown event per mouse click!

Example1