

# Wir erstellen unsere eigene App!

1. Tools installieren (Android plugin, Android sdk, Sprache Deutsch)
  2. Mit „gdx-liftoff.jar“ ein Projekt erstellen und öffnen
- „core“ enthält unseren Java-Code.
  - „android“ lässt unser Spiel auf Android-Geräten laufen.
  - „lwjgl3“ lässt unser Spiel auf Computern laufen.
  - „assets“ enthält Grafiken, Sounds und Texte.

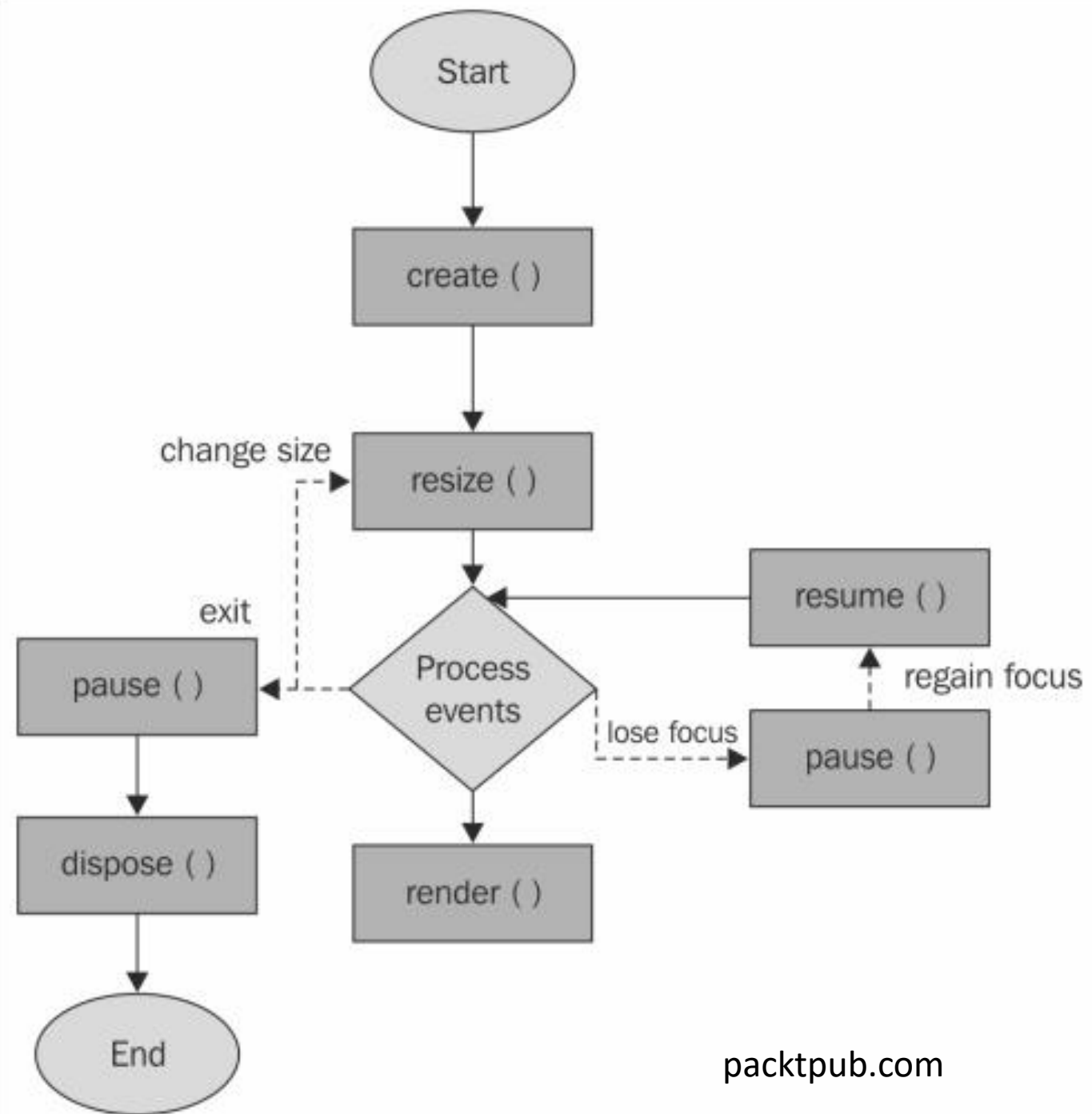


# Android debug bridge

- Öffnet die Einstellungen eures Android-Handys.
- Klickt auf „Über dieses Gerät“.
- Findet im Menü „Build-Nummer“, „OS version“, „MIUI version“ bzw. „Software version“.
- Klickt ganz oft auf diesen Menüpunkt. (wirklich!)
- Sucht nach „Entwickleroptionen“.
- Scrollt runter und schaltet „USB-Debugging“, „Installieren über USB“ an.
- Schließt euer Handy an den Computer an.
- Klickt in IntelliJ auf die Auswahl links neben dem grünen Startknopf und wählt „android“ aus.
- Überprüft, ob euer Handy verbunden ist, startet das Spiel und bestätigt die Installation auf eurem Handy.
- Es sollte sich eine App mit libGDX-Logo auf dem Handy öffnen.

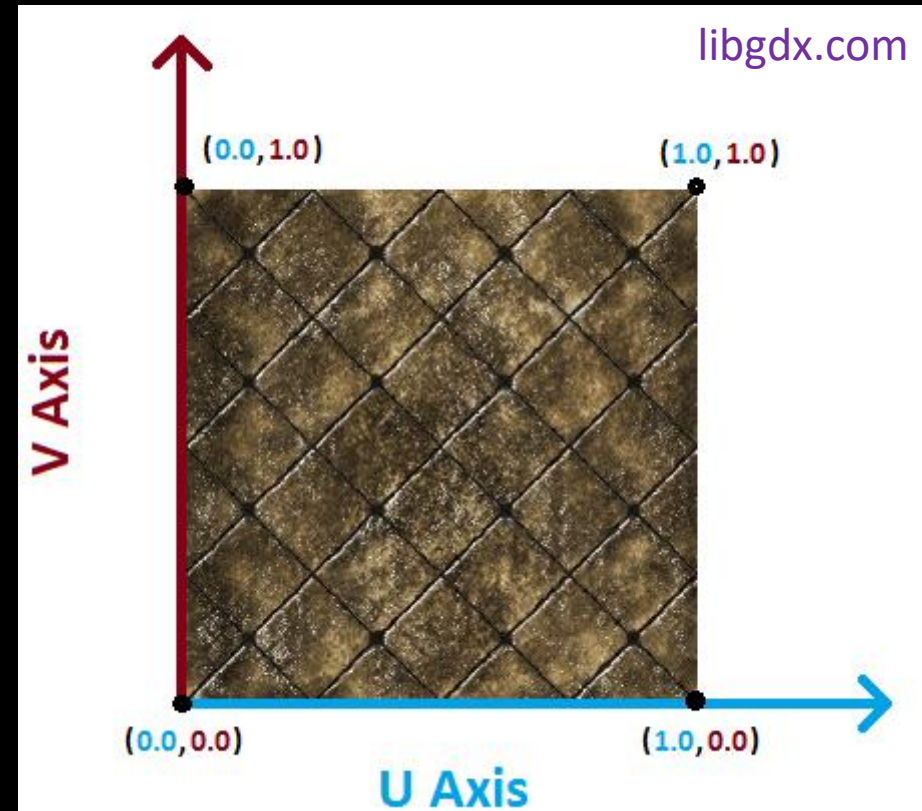
# Lebenszyklus

Verschiedene Fälle, in denen wir Code ausführen können.



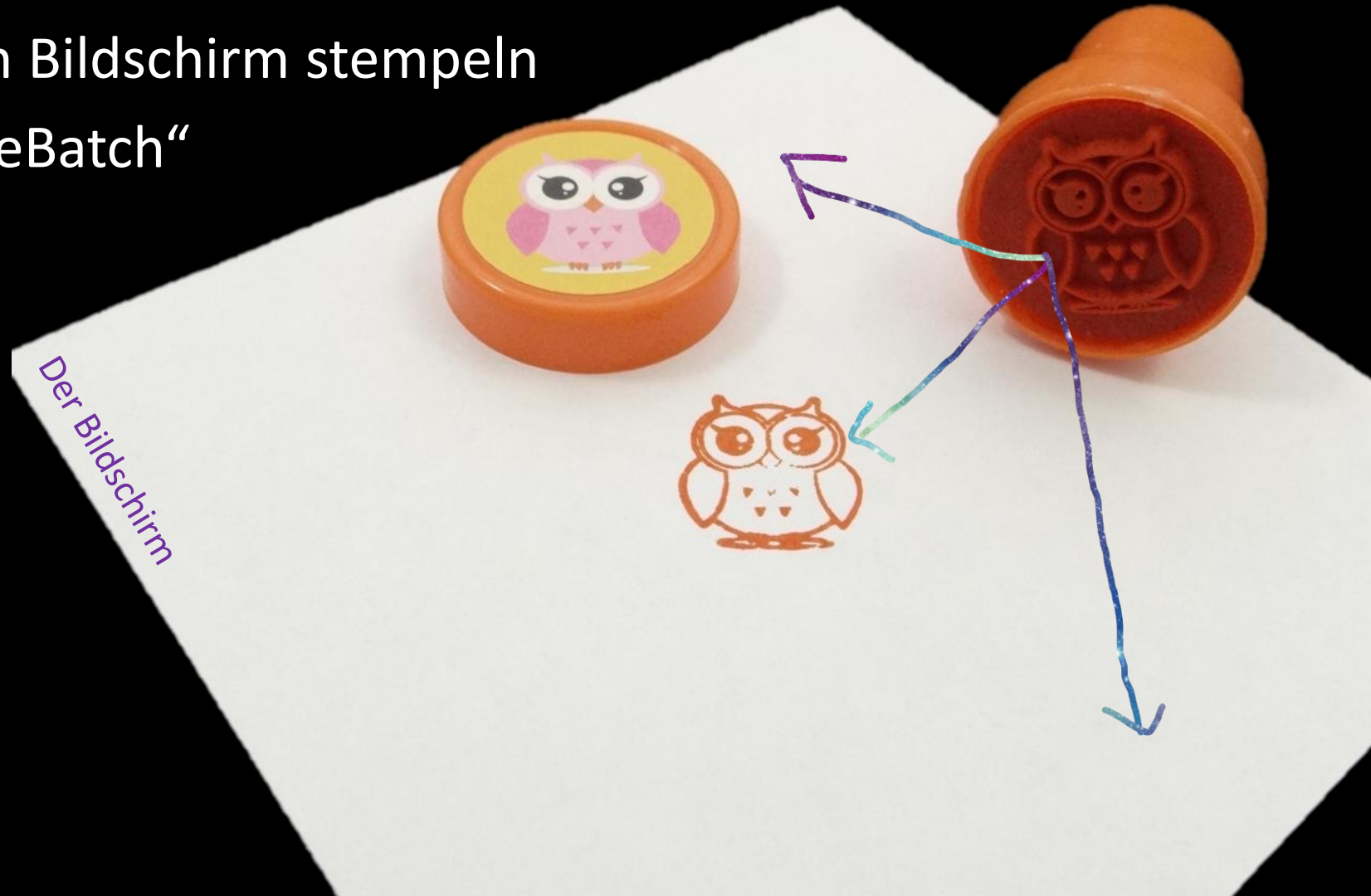
# Texture

- Ein Bild, das im Spiel dargestellt werden kann



# Batch

- Kann Texturen auf den Bildschirm stempeln
- Wir verwenden „SpriteBatch“



# Instanzmethoden – Batch

Name	Parameter	Bedeutung
begin		Bereitet den Batch auf das Merken der Positionen vor.
end		ruft „flush()“ auf, um die Textur an allen gemerkten Positionen auf den Bildschirm zu malen.
Es gibt mehrere draw-Methoden, z.B.:		
draw	Texture texture, float x, float y, float width, float height	Lässt die Textur bei x und y mit festgelegter Breite und Höhe malen.
(draw)	Texture texture, float x, float y, float originX, float originY, float width, float height, float scaleX, float scaleY, float rotation, float srcX, float srcY, int srcWidth, int srcHeight, boolean flipX, boolean flipY	Nimmt den rechteckigen Ausschnitt bei srcX / srcY mit srcWidth und srcHeight von texture, skaliert und dreht ihn um originX und originY, dreht ihn ggf. um und malt ihn bei der Position x und y mit Breite width und Höhe height auf den Bildschirm.

# ShapeRenderer

- Kann geometrische Formen auf den Bildschirm malen



# Instanzmethoden – ShapeRenderer

Name	Parameter	Bedeutung
begin	ShapeType type	Bereitet den ShapeRenderer auf das Zeichnen der Formen vor.
	ShapeRenderer.ShapeType.Line	Zeichnet den Umriss von Formen.
	ShapeRenderer.ShapeType.Filled	Zeichnet Formen komplett aus.
end		Zeichnet alle gezeichneten Formen auf den Bildschirm.
circle	float x, float y, float radius (, int segments)	Zeichnet den Kreis bei x und y mit Radius radius durch Kombination vieler kleiner Striche (Vieleck). Tipp: Übergebe dazu noch eine Anzahl an Strichen, falls der Kreis zu eckig ist.
rect (Abkürzung für „rectangle“)	float x, float y, float width, float height	Zeichnet ein Rechteck bei x und y mit Breite width und Höhe height



# Gdx.input

- Kann Eingaben einlesen



# Instanzmethoden – Input

Name	Parameter	Bedeutung
getX	(int pointer)	x-Koordinate des Fingers bzw. Mauszeigers. Bei mehreren Fingern kann man durch Übergeben des Pointers nach einem bestimmten Finger fragen.
getY	^^	y-Koordinate ^^

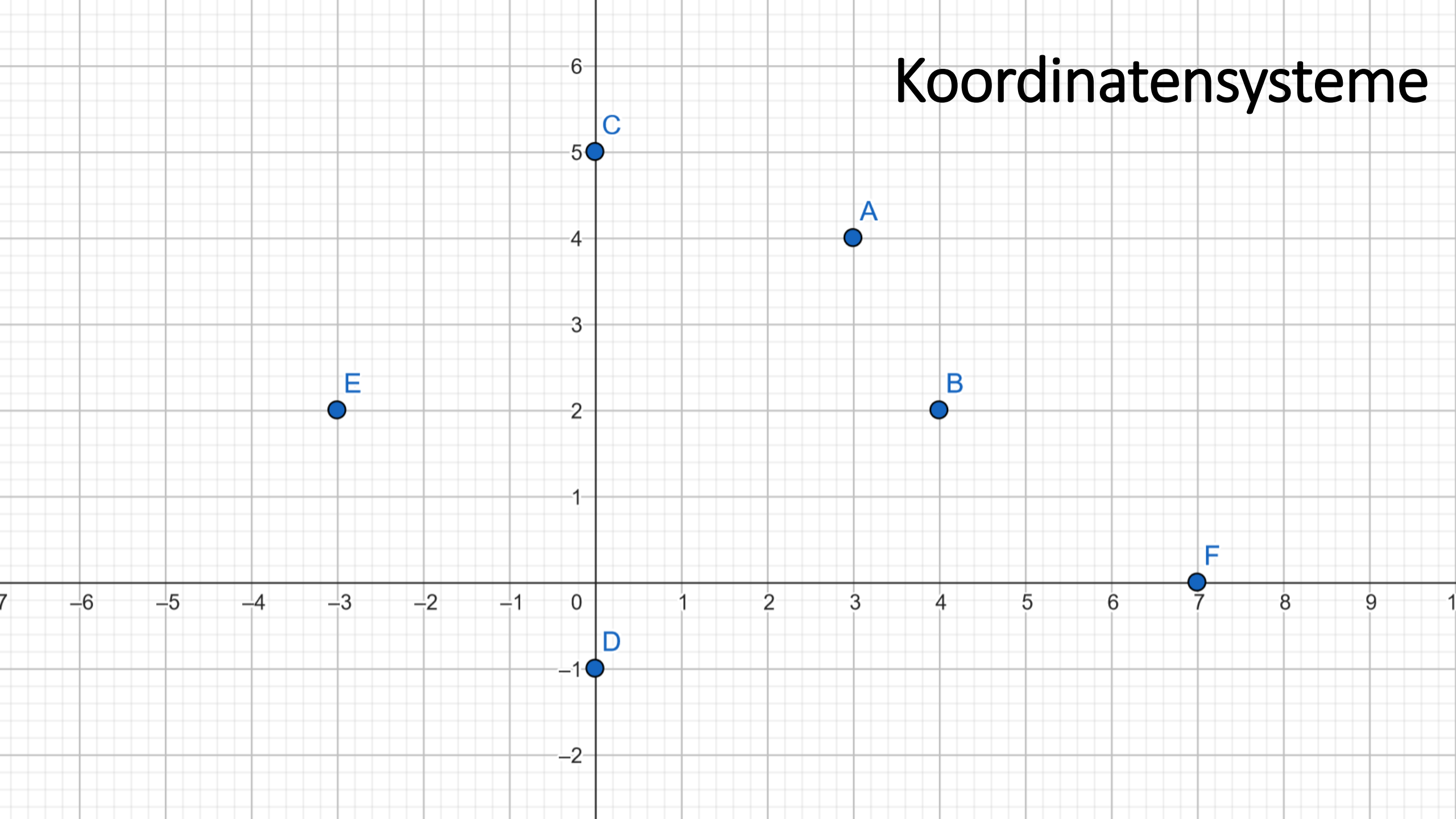
# ScreenUtils.clear

- Füllt den gesamten Bildschirm mit einer Farbe aus.
- Übermalt alles, was sich vorher auf dem Bildschirm befunden hat.
- Die Parameter r, g und b legen die Farbkomponenten (rot, grün, blau) der Farbe fest.
  - Gut, um eine (sich verändernde?) Hintergrundfarbe festzulegen.

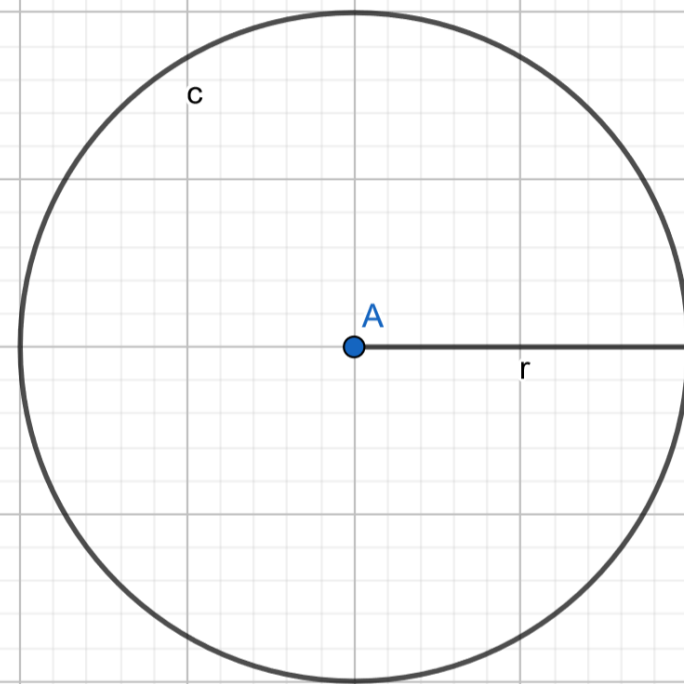
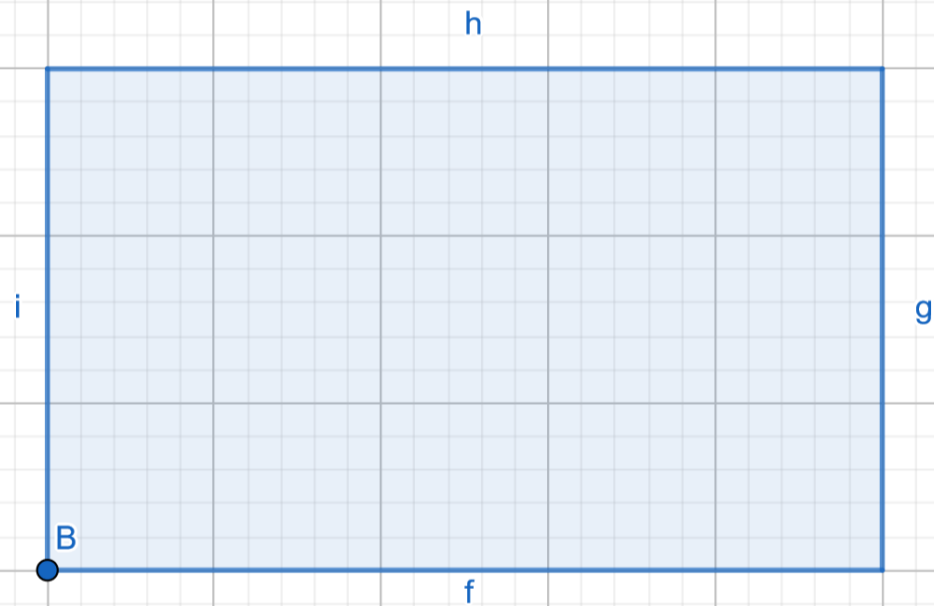
# Wir wenden das Wissen an.

```
@Override
public void render() {
    ScreenUtils.clear(0.15f, 0.15f, 0.2f, 1f);
    batch.begin();
    int x = Gdx.input.getX();
    int y = Gdx.input.getY();
    batch.draw(image, x, y);
    batch.end();
}
```

# Koordinatensysteme



# Formen



# Probleme beheben

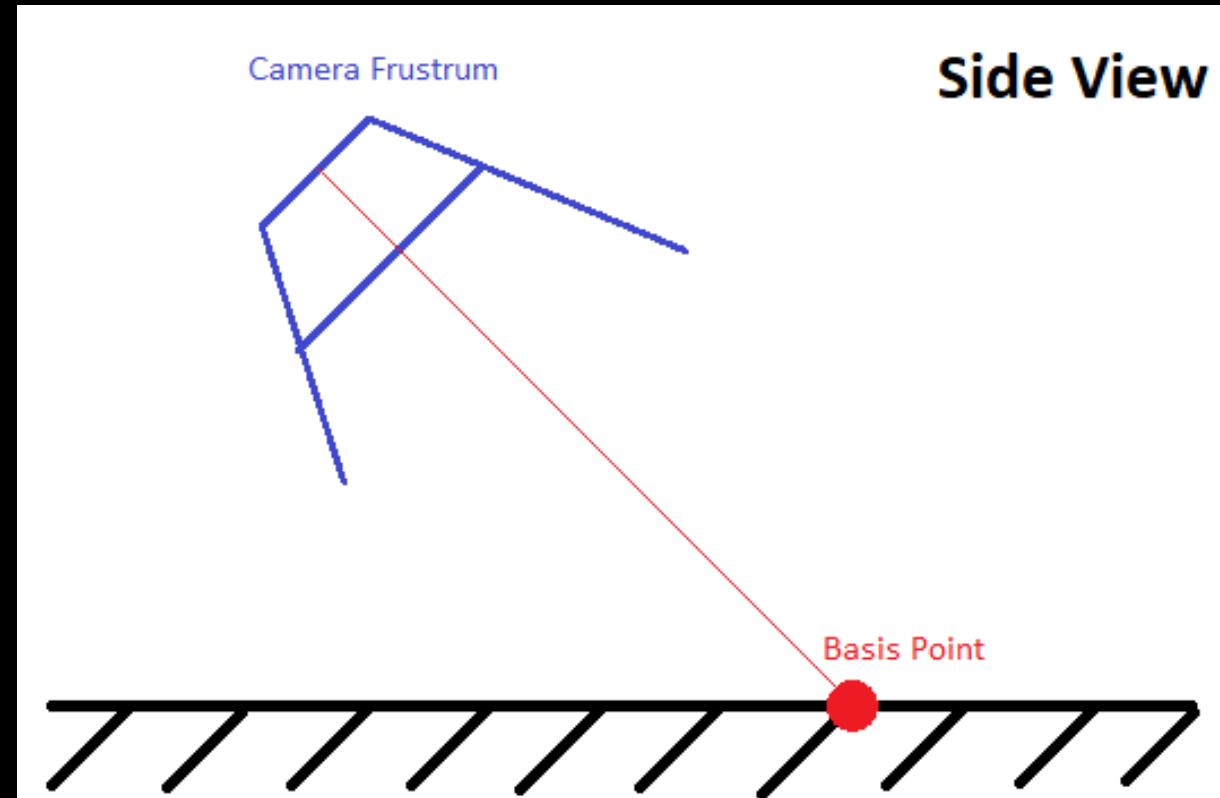
```
private void maleMittig(Texture texture, int x, int y) {  
    batch.draw(texture,  
        x - texture.getWidth() / 2f,  
        y - texture.getHeight() / 2f);  
}
```

@Override

```
public void render() {  
    ScreenUtils.clear(0.15f, 0.15f, 0.2f, 1f);  
    batch.begin();  
    int x = Gdx.input.getX();  
    int y = Gdx.graphics.getHeight() - Gdx.input.getY();  
    maleMittig(image, x, y);  
    batch.end();  
}
```

# Camera

- Kennt die dargestellte Position auf dem Spielfeld
- Kann die Spielwelt auf die Bildschirmwelt und zurück projizieren
- OrthographicCamera
  - Für 2D-Spiele
  - Kennt den Zoom





# Instanzmethoden - OrthographicCamera

Name	Parameter	Bedeutung
translate	float x, float y, float z	Bewegt die Kamera um x in x-Richtung und um y in y-Richtung. Die z-Koordinate ist für 2D-Spiele egal.
rotate	float angle	Dreht die Kamera um angle Grad, normalerweise gegen den Uhrzeigersinn.
update		Aktualisiert die Projektionsmatrix der Kamera nach Änderungen.
project	Vector3 worldCoords	Wandelt Welt-Koordinaten in Bildschirm-Koordinaten um.
unproject	Vector3 screenCoords	Wandelt Bildschirm-Koordinaten in Welt-Koordinaten um

# OrthographicCamera - zoomen

Die OrthographicCamera beinhaltet das Attribut „zoom“. Ihr könnt es verändern, da es den Sichtbarkeitsmodifikator „public“ hat:

```
camera.zoom = 1.5f;  
camera.update();
```

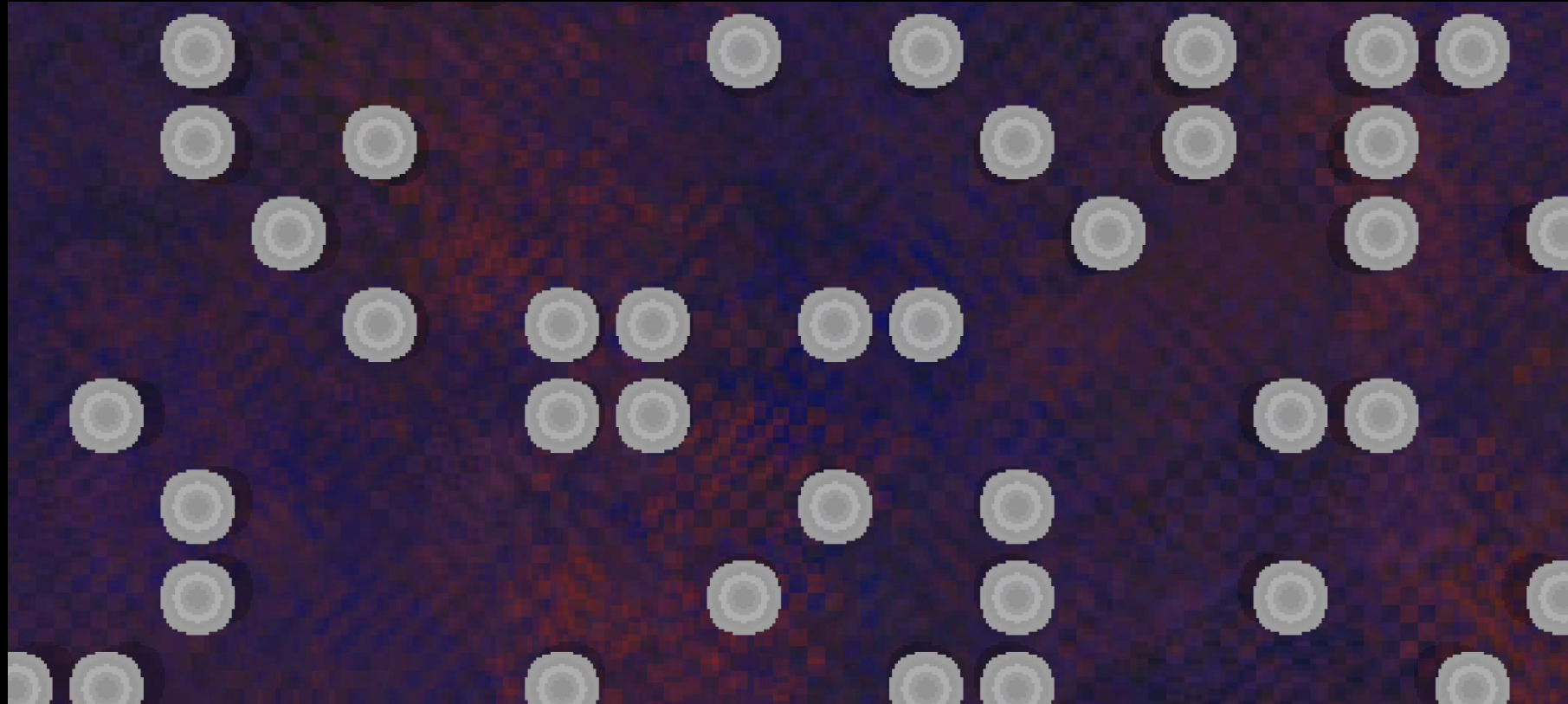
# Kamera verwenden

Die Position, Skalierung und Drehung der Kamera sind in der kombinierten Matrix gespeichert. Man kann über das Attribut „combined“ auf diese zugreifen und über die Methode „setProjectionMatrix“ auf SpriteBatch bzw. ShapeRenderer anwenden:

```
batch.setProjectionMatrix(camera.combined);  
renderer.setProjectionMatrix(camera.combined);
```

# Viewport

- Sagt der Kamera, wie sie die Spielwelt auf den Bildschirm bringen soll.
- `ExtendViewport`
  - Erweitert den Ausschnitt der Welt, um den Bildschirm auszufüllen



# ExtendViewport verwenden

- ExtendViewport muss drei Dinge wissen:
  - Die minimale Anzahl an Welteinheiten in x- und y-Richtung
  - Die Kamera, um dieser die Informationen mitzuteilen
  - Wie groß das Fenster zu jedem Zeitpunkt ist

```
private final Viewport viewport = new ExtendViewport(10, 10, camera);
```

```
@Override
```

```
public void resize(int width, int height) {
```

```
    viewport.update(width, height);
```

```
}
```

Welteinheiten in x-Richtung

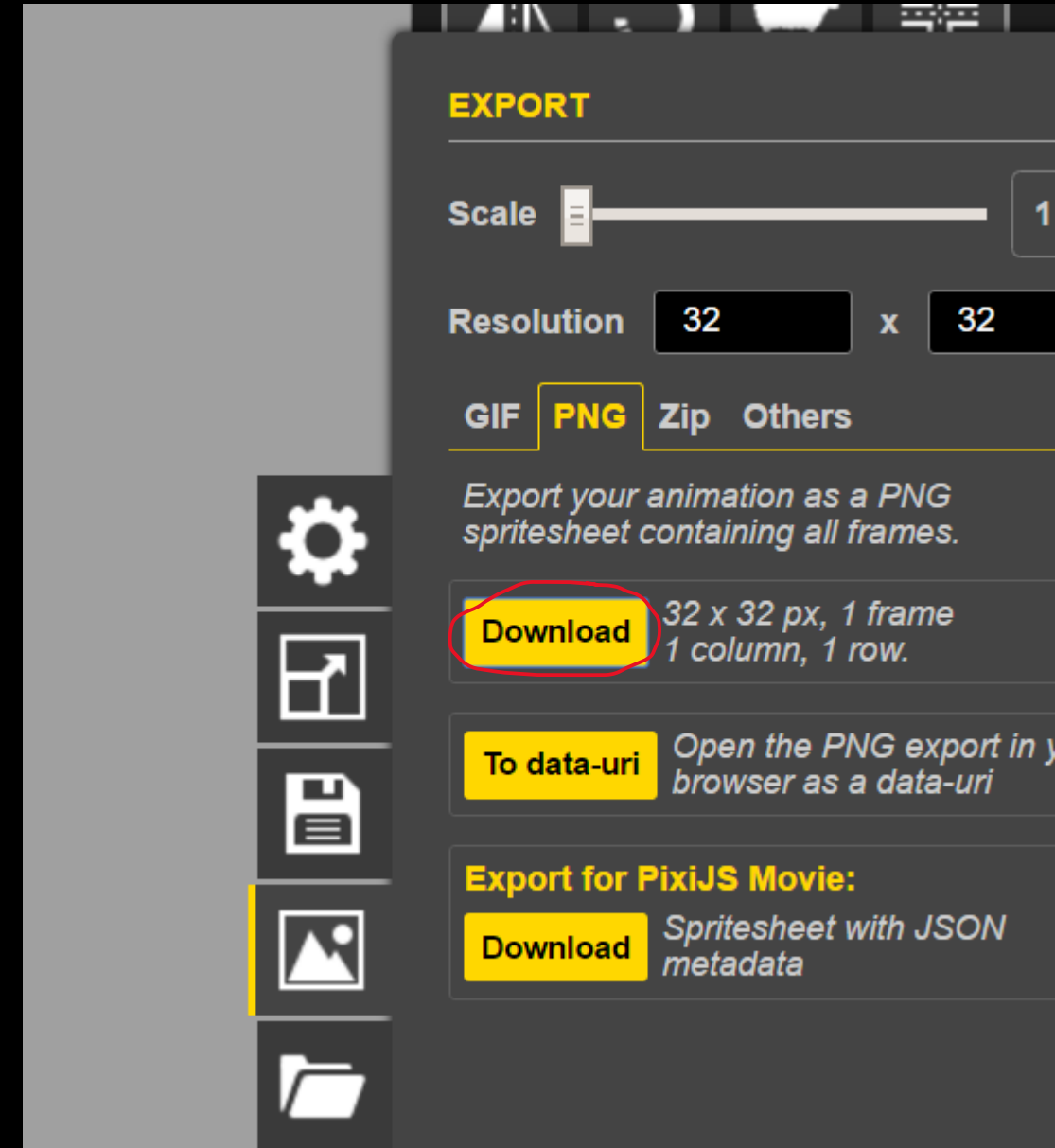
Welteinheiten in y-Richtung

Die Kamera

Größe des Fensters, wenn diese sich ändert

# Eigene Texturen

- Malt eure Grafiken in Piskel.
- Speichert sie als .png-Datei auf dem Desktop.
- Zieht sie in den assets-Ordner.
- Gebt beim Erstellen der Texture den Dateinamen an (z.B. "affe.png").



# Sprite

...

# maleMittig – mit Größe

```
private void maleMittig(Texture texture, float x, float y, float width) {  
    float height = width * texture.getHeight() / texture.getWidth();  
    batch.draw(texture, x - width / 2, y - height / 2, width, height);  
}
```

Wir übergeben nur die Breite. Die Höhe wird automatisch berechnet, indem wir die Breite mit der Auflösung der Textur (Höhe der Textur durch Breite der Textur) multiplizieren.



# Möglichkeit 1 – viele Variablen für Texturen

```
private Texture
    character, tree, grass, axe, water;

@Override
public void create() {
    // sonstiger Code ...
    character = new Texture("character.png");
    tree = new Texture("tree.png");
    grass = new Texture("grass.png");
    axe = new Texture("axe.png");
    water = new Texture("water.png");
}
```

```
@Override
public void render() {
    float cameraX = camera.position.x, cameraY = camera.position.y;
    maleMittig(character, cameraX, cameraY, 1);
    maleMittig(tree, 4, 4, 3);
    for (int i = 1; i < 10; i++) maleMittig(grass, i, 1, 1);
    maleMittig(axe, cameraX, cameraY + 1, 1);
    for (int i = -5; i < 5; i++) {
        maleMittig(water, -1, i, 2);
        if (Math.abs(i) != 5) maleMittig(water, -3, i, 2);
    }
}
```

# Möglichkeit 2 – AssetManager

- Wir können den AssetManager verwenden, um all unsere assets zu verwalten:

```
private final AssetManager manager = new AssetManager();

private void maleMittig(String name, float x, float y, float width) {
    String fileName = name + ".png";
    manager.load(fileName, Texture.class);
    manager.finishLoading();
    Texture texture = manager.get(fileName);
    float height = width * texture.getHeight() / texture.getWidth();
    batch.draw(texture, x - width / 2, y - height / 2, width, height);
}
```

# Aufgabe

- Platziert mit Hilfe der maleMittig-Methode und dem ShapeRenderer verschiedene Formen in die Welt.

# Kamera bewegen (Mathe)

```
if (Gdx.input.isTouched()) { // Wenn wir auf den Bildschirm drücken
    int x = Gdx.input.getX(); // x-Koordinate des Fingers/Mauszeigers
    int y = Gdx.input.getY(); // y-Koordinate des Fingers/Mauszeigers
    // Rechne aus, wo auf dem Spielfeld wir hinklicken
    Vector3 spielweltKoordinaten = camera.unproject(new Vector3(x, y, 0));
    float geschwindigkeit = 6; // Geschwindigkeit in Welteinheiten pro Sekunde
    Vector3 bewegung = new Vector3(spielweltKoordinaten);
    // Richtungsvektor von Mitte des Spielfelds zu Finger/Mauszeiger
    bewegung.sub(camera.position);
    // Berechne den Geschwindigkeitsvektor durch Skalierung auf geschwindigkeit Welteinheiten.
    bewegung.scl(geschwindigkeit / bewegung.len());
    // Multipliziere den Vektor mit der Zeit (Strecke = Geschwindigkeit * Zeit).
    bewegung.scl(Gdx.graphics.getDeltaTime());
    camera.translate(bewegung);
}
```

# Überlappung

```
public boolean collidesWith(Entity other) {  
    float maximalerUnterschiedInX = getWidth() / 2 + other.getWidth() / 2;  
    float maximalerUnterschiedInY = getHeight() / 2 + other.getHeight() / 2;  
    return Math.abs(x - other.x) < maximalerUnterschiedInX  
        && Math.abs(y - other.y) < maximalerUnterschiedInY;  
}
```

Die Methode setzt voraus, dass eine Entity-Klasse mit Attributen x, y und Methoden getWidth() und getHeight() besteht.