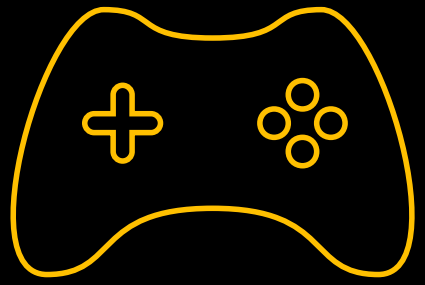

Bonus
Theorie in praktischen Projekten anwenden

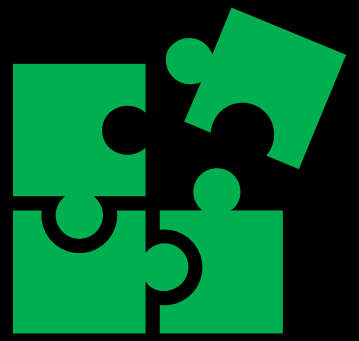
# Code Schnipsel

Die lilafarben markierten Codeteile musst du selbst ausfüllen.

Erstelle alle Klassen in `core/src/main/java/root.content`.
Schreibe die Klassennamen richtig und in UpperCamelCase.

Melde dich bitte bei allen Schwierigkeiten.





```java
public class Colors {
    // Die Farben bestehen aus red, green, blue und alpha (= Opazität).
    // Opazität: 0f = unsichtbar, 0.5f = semi transparent, 1f = vollständig sichtbar.
    // Alle Komponenten werden zwischen 0f und 1f angegeben.
    // Importiere com.badlogic.gdx.graphics.Color, nicht java.awt.Color.
    public static final Color
        POSITIVE_COLOR = new Color(/* ??? */, /* ??? */, /* ??? */, /* ??? */),
        NEGATIVE_COLOR = new Color(/* ??? */, /* ??? */, /* ??? */, /* ??? */),
        NEUTRAL_COLOR = new Color(/* ??? */, /* ??? */, /* ??? */, /* ??? */),
        HEALTH_BAR_COLOR = new Color(/* ??? */, /* ??? */, /* ??? */, /* ??? */);
}

/**
 * Repräsentiert die Koordinaten eines Ortsvektors der Spielwelt.
 */
public record Position(int x, int y) {

    public Position add(Position other) {
        return new Position((/* ??? */, /* ??? */);
    }
}
```

```java
public enum Direction {
    UP(new Position(0, 1), Input.Keys.W),
    DOWN(/* ??? */, Input.Keys.S),
    LEFT(new Position(-1, 0), /* ??? */),
    RIGHT(/* ??? */, Input.Keys.D);

    private final Position position;
    private final int key;

    Direction(Position position, int key) {
        this.position = position;
        // ??? (initialisiere this.key)
    }

    public int getKey() {
        return key;
    }

    // ??? (getter für position)
}
```

```java
public enum Field {
    NEUTRAL(0),
    ONE_POSITIVE(1), /* ??? */, /* ??? */, FOUR_POSITIVE(4),
    /* ??? */, TWO_NEGATIVE(-2), /* ??? */, /* ??? */;

    private final String textureName;
    private final ??? color;
    private final int points;

    Field(int points) {
        this.textureName = /* ??? (Format: dot<Absolute Punktanzahl>.png) */;
        // ??? (initialisiere this.points)
        this.color = points < 0 ? Colors.NEGATIVE_COLOR : /* ??? (zweiter Ternary) */;
    }

    // ??? (getter für textureName, color und points)
}
```

```java
public class World {
    private final HashMap<Position, Field> fields = new HashMap<>();
    private final Random random = new Random();
    private Position playerPosition = new Position(0, 0);
    private int points = 100;

    public Field getField(Position position) {
        Field field = /* ??? (Hole das Feld an position aus der HashMap fields.) */;
        if (field == null) {
            Field[] allFields = Field.values();
            // Tipp: Schau dir die Aufgabe "Lustige Sätze" an.
            field = /* ??? (Wähle ein zufälliges Feld aus.) */;
            // ??? (Schreibe das neue Feld an position in die HashMap fields.)
        }
        return field;
    }

    private void move(Direction direction) {
        // ??? (Zuletzt implementieren: Was muss passieren, wenn man sich bewegt?)
    }
}
```

# World fertigstellen

Ergänze `World` um folgende Methoden:

- Methode `isPlayerAlive` (Gibt es noch Punkte?)

```
public void update() {
    if (!isPlayerAlive()) return;
    for (Direction direction : Direction.values())
        if (Gdx.input.isKeyJustPressed(direction.getKey()))
            move(direction);
}
```

- getter für `playerPosition` und `points`

# Klasse Renderer

Die Java-Klasse `Renderer` stellt die Welt auf dem Bildschirm da.

Weil sie sehr lang ist und nicht viele Erkenntnisse bietet, haben wir uns entschieden, sie euch zum Kopieren zu geben.

Die zugehörigen Assets musst du auch herunterladen und in den assets-Ordner (resources root) kopieren.

```java
public class Main extends ApplicationAdapter {
    private final World world = new World();
    private Renderer renderer;

    @Override
    public void create() {
        // ??? (Initialisiere renderer durch ein neues Objekt.)
    }

    @Override
    public void resize(int width, int height) {
        // ??? (Rufe die resize-Methode von renderer auf.)
    }

    @Override
    public void render() {
        world.update();
        // ??? (Verwende renderer, um die Welt darzustellen.)
    }
}
```

# 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

- Öffne die Einstellungen deines Android-Handys.
- Klicke auf „Über dieses Gerät“.
- Suche im Menü „Build-Nummer“, „OS version“, „MIUI version“ bzw. „Software version“.
- Klicke ganz oft auf diesen Menüpunkt. (wirklich!)

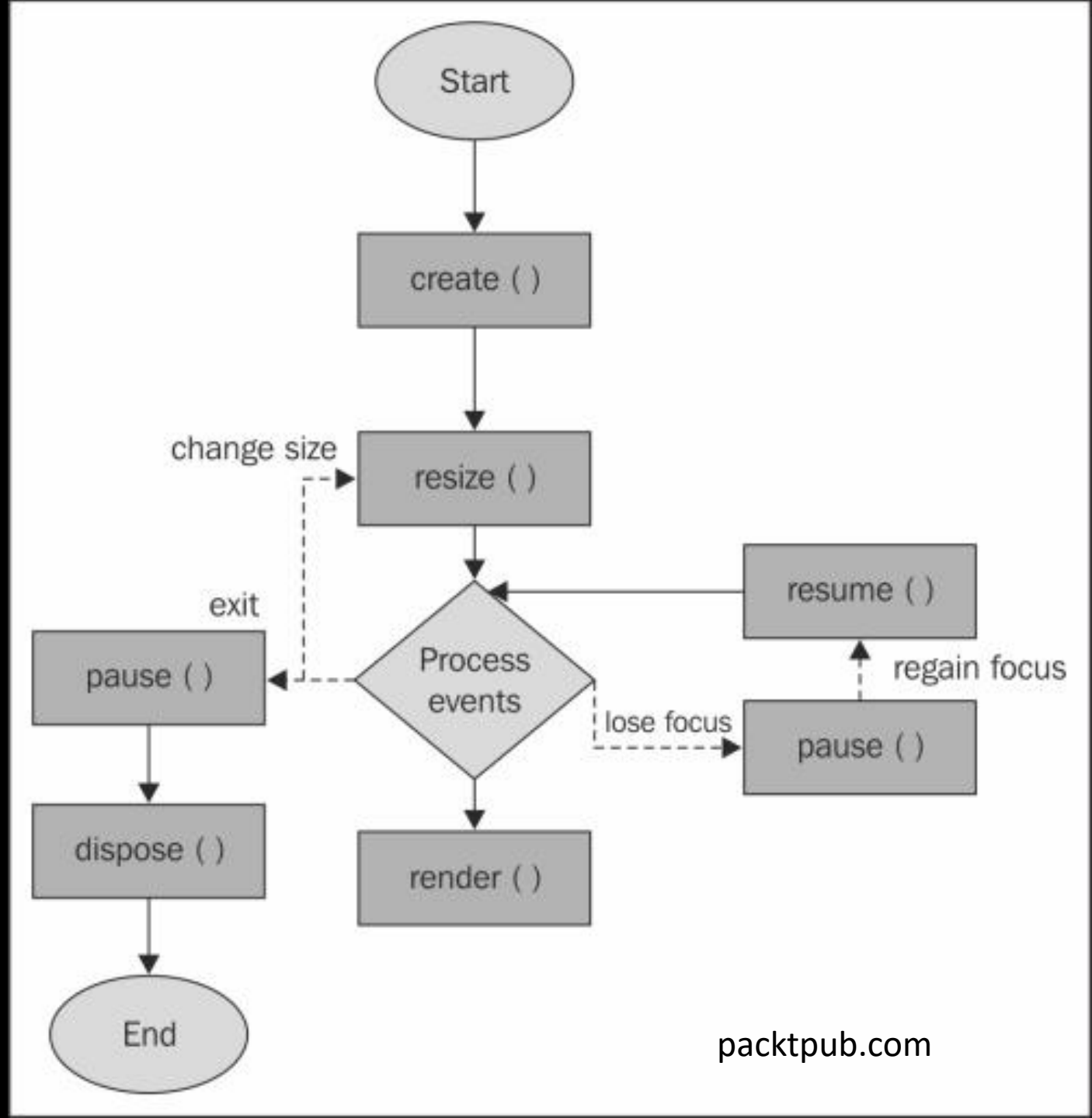
Es erscheint die Toast-Nachricht "Okay, du bist bereits Entwickler."

- Suche nach „Entwickleroptionen“.
- Scroll runter und schalte „USB-Debugging“ und, falls existent, „Installieren über USB“ an.
- Schließ dein Handy an den Computer an.
- Klicke in IntelliJ auf die Konfigurationsauswahl links neben dem grünen Startknopf und wähle „android“ aus.
- Überprüfe, ob der Gerätename deines Handys links daneben angezeigt wird
- Starte das Spiel, warte und bestätige die Installation auf deinem Handy.

Es wird eine App mit libGDX-Logo installiert und automatisch geöffnet.

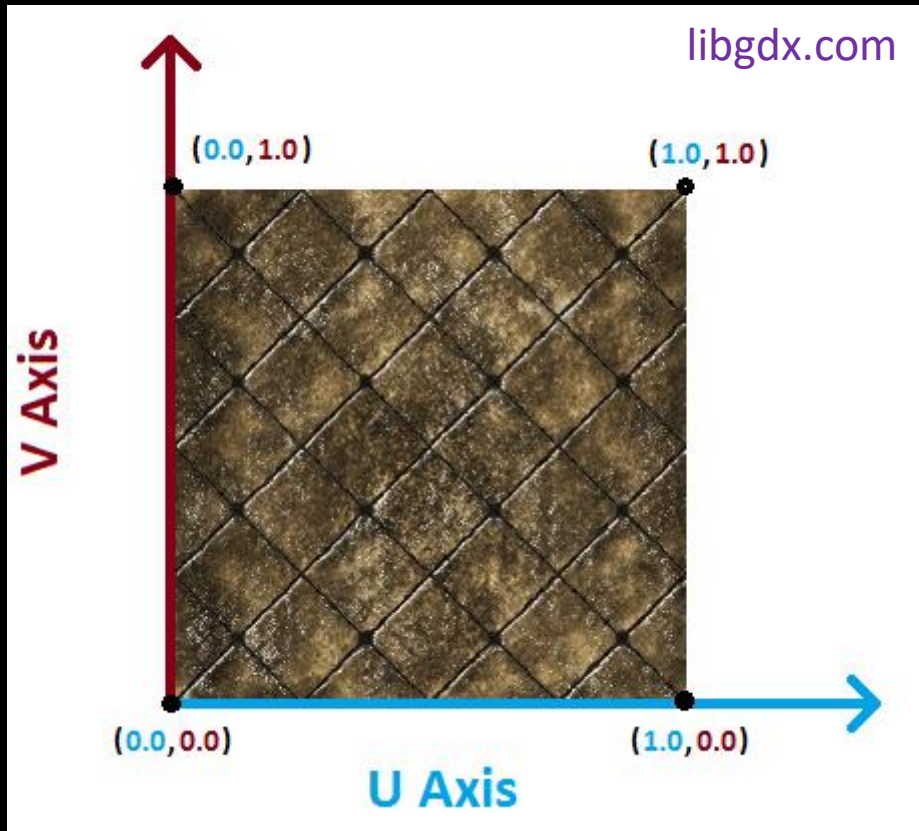
# Lebenszyklus

Verschiedene Fälle, in denen wir Code ausführen können.



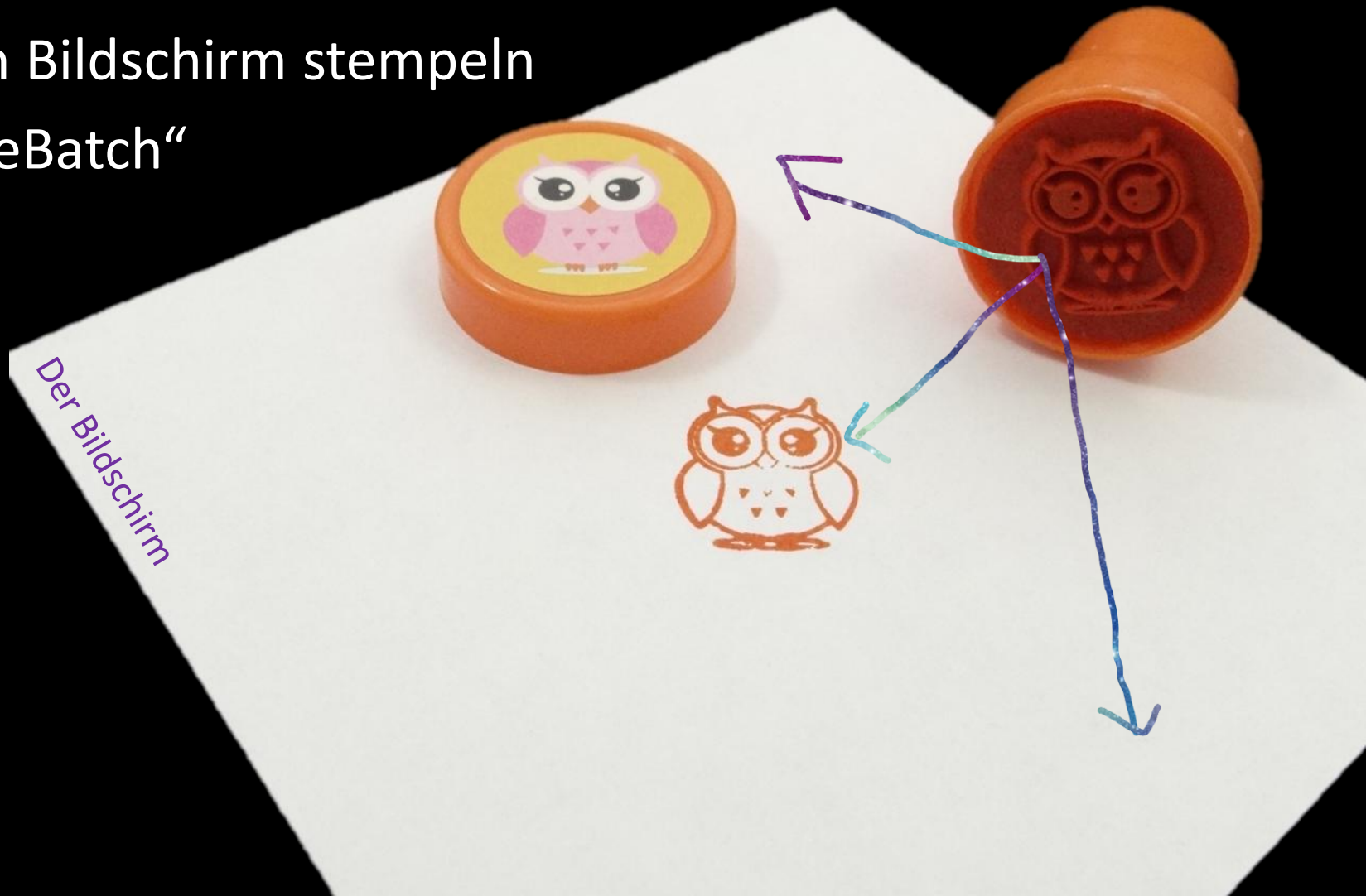
# Texture

- 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 Merken der Formen vor. |
| | ShapeRenderer.ShapeType.Line | Malt den Umriss von Formen. |
| | ShapeRenderer.ShapeType.Filled | Malt Formen komplett aus. |
| end | | Malt alle gemerkten Formen auf den Bildschirm. |
| circle | float x, float y, float radius<br>(, int segments) | Malt den Kreis bei x und y mit Radius radius durch Kombination vieler kleiner Striche (Vieleck).<br>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 | Malt 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.<br>Bei mehreren Fingern kann man durch Übergeben des Pointers nach einem bestimmten Finger fragen. |
| getY | ^^ | y-Koordinate ^^ |

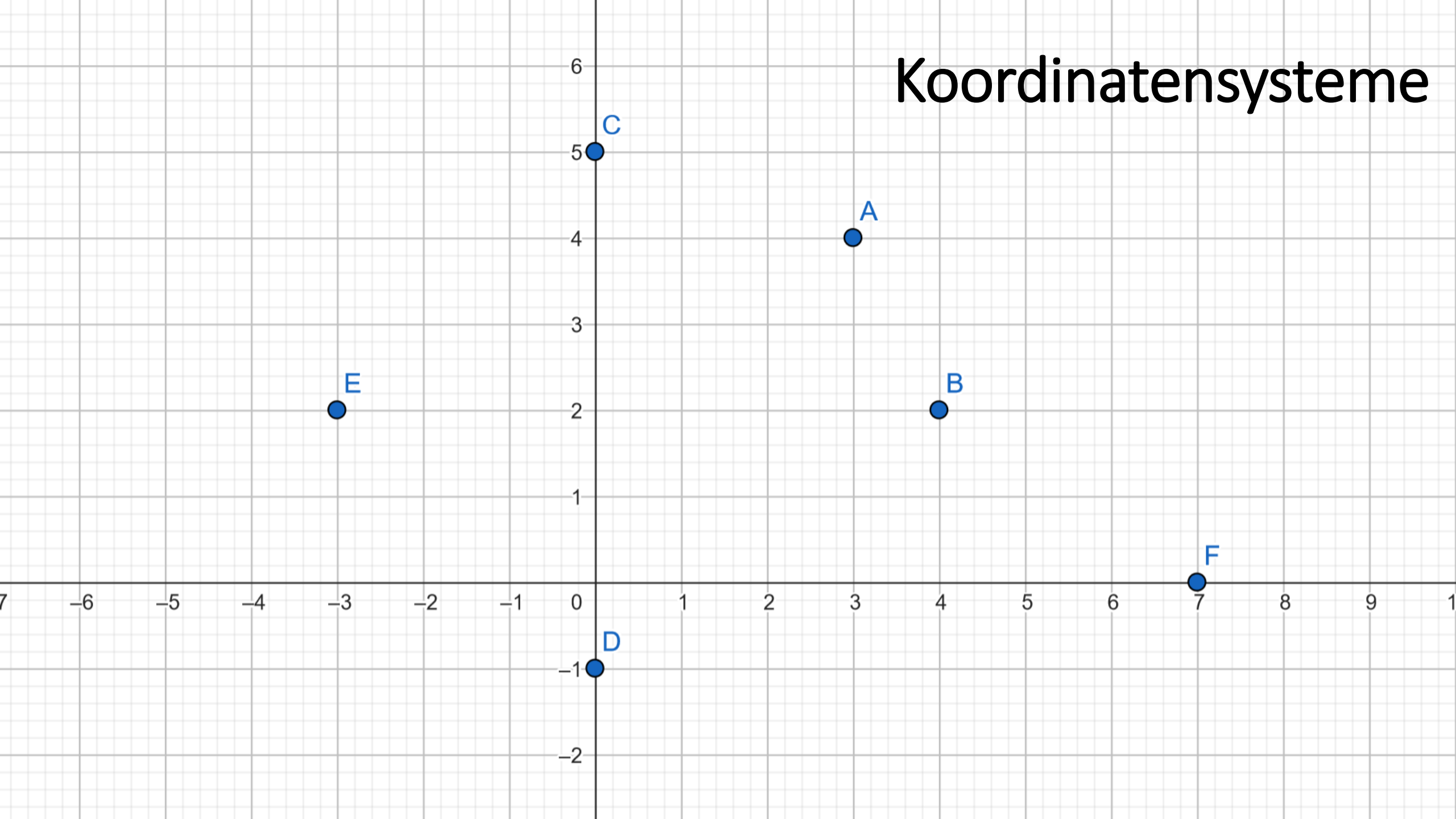
# ScreenUtils.clear

- 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.

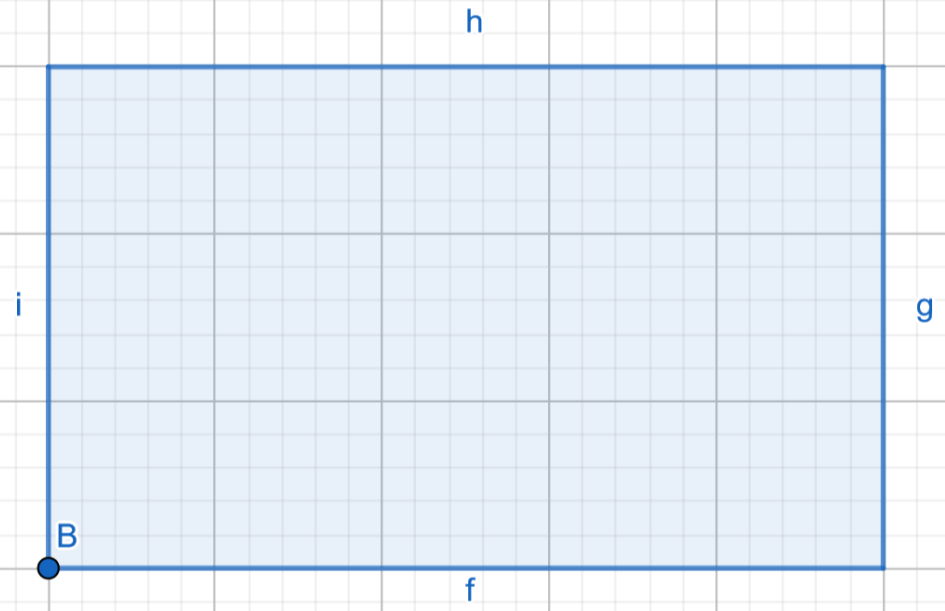
# Wir wenden das Wissen an.

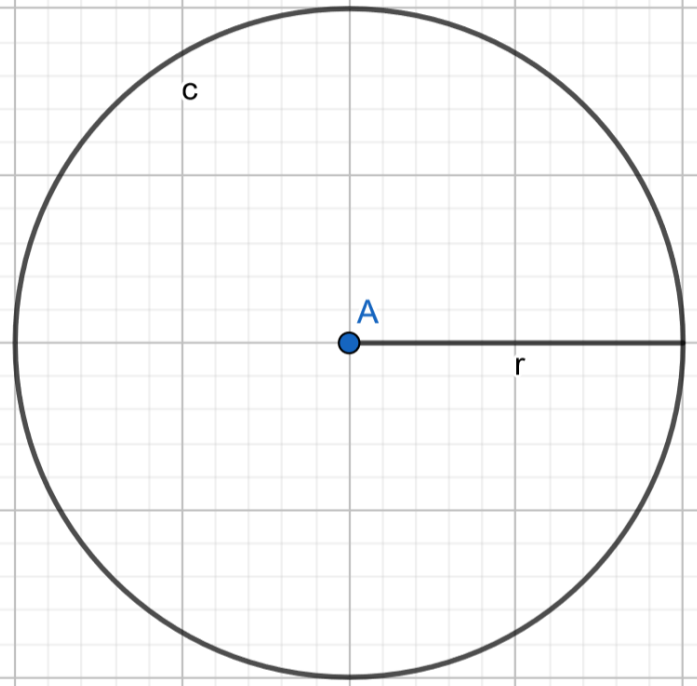
```
@Override
public void render() {
    ScreenUtils.clear(0.15f, 0.15f, 0.2f, 1f);
    batch.begin();
    int x = Gdx.input.getX();
    int y = Gdx.input.getY();
    batch.draw(image, x, y);
    batch.end();
}
```

# Koordinatensysteme



# Formen



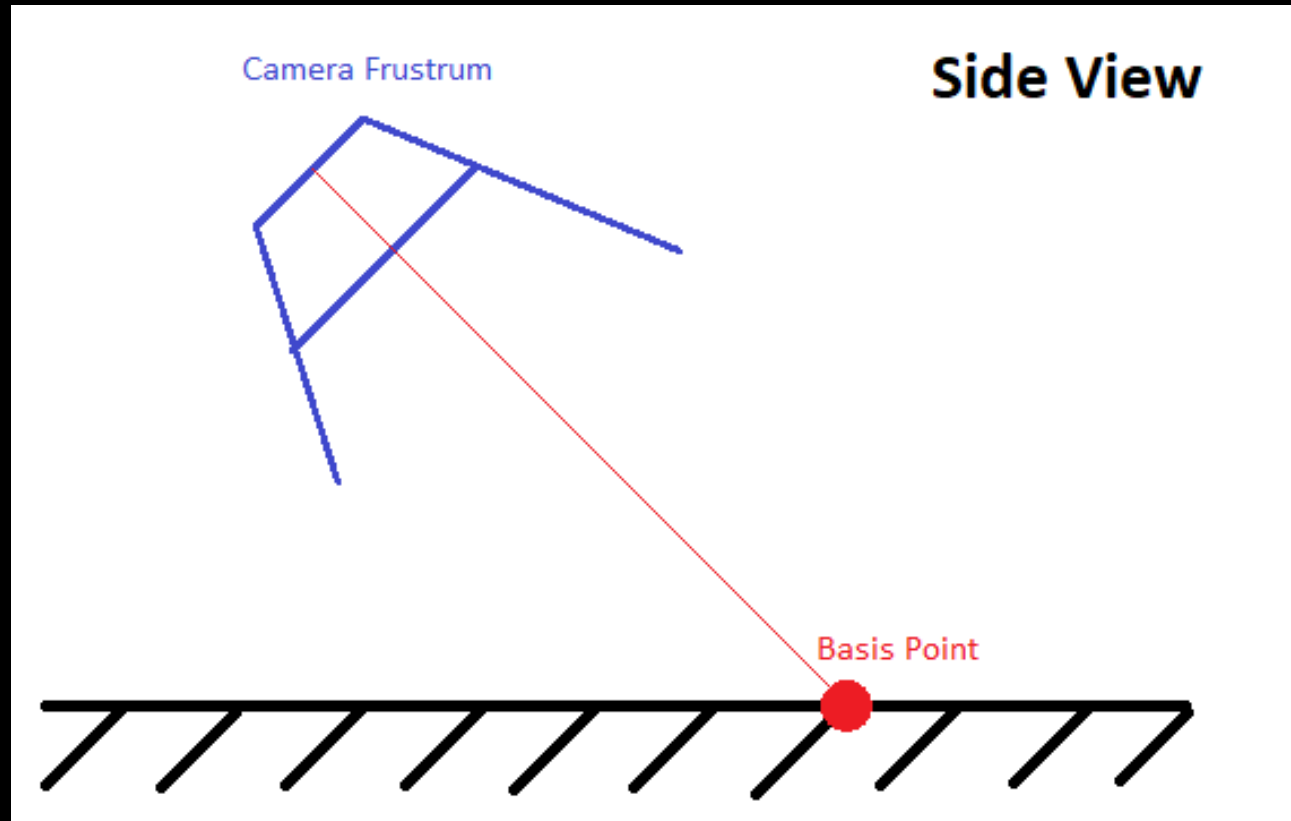
```
private void maleMittig(Texture texture, int x, int y) {
    batch.draw(texture,
        x - texture.getWidth() / 2f,
        y - texture.getHeight() / 2f);
}

@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();
}
```

Probleme beheben

# 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

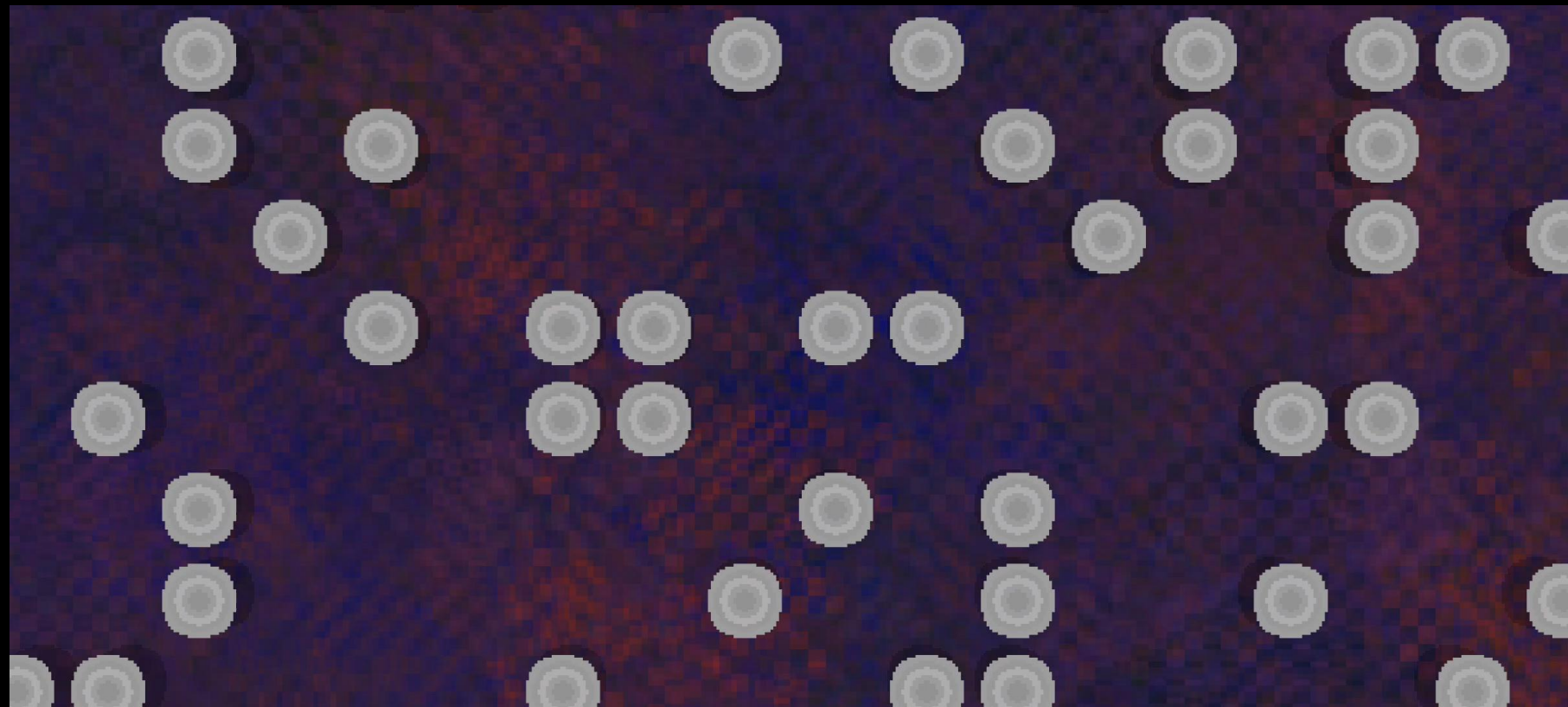
Die Position, Skalierung und Drehung der Kamera sind in der kombinierten Matrix gespeichert. Man kann über das Attribut „combined“ auf diese zugreifen und über die Methode „setProjectionMatrix“ auf SpriteBatch bzw. ShapeRenderer anwenden:

```
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

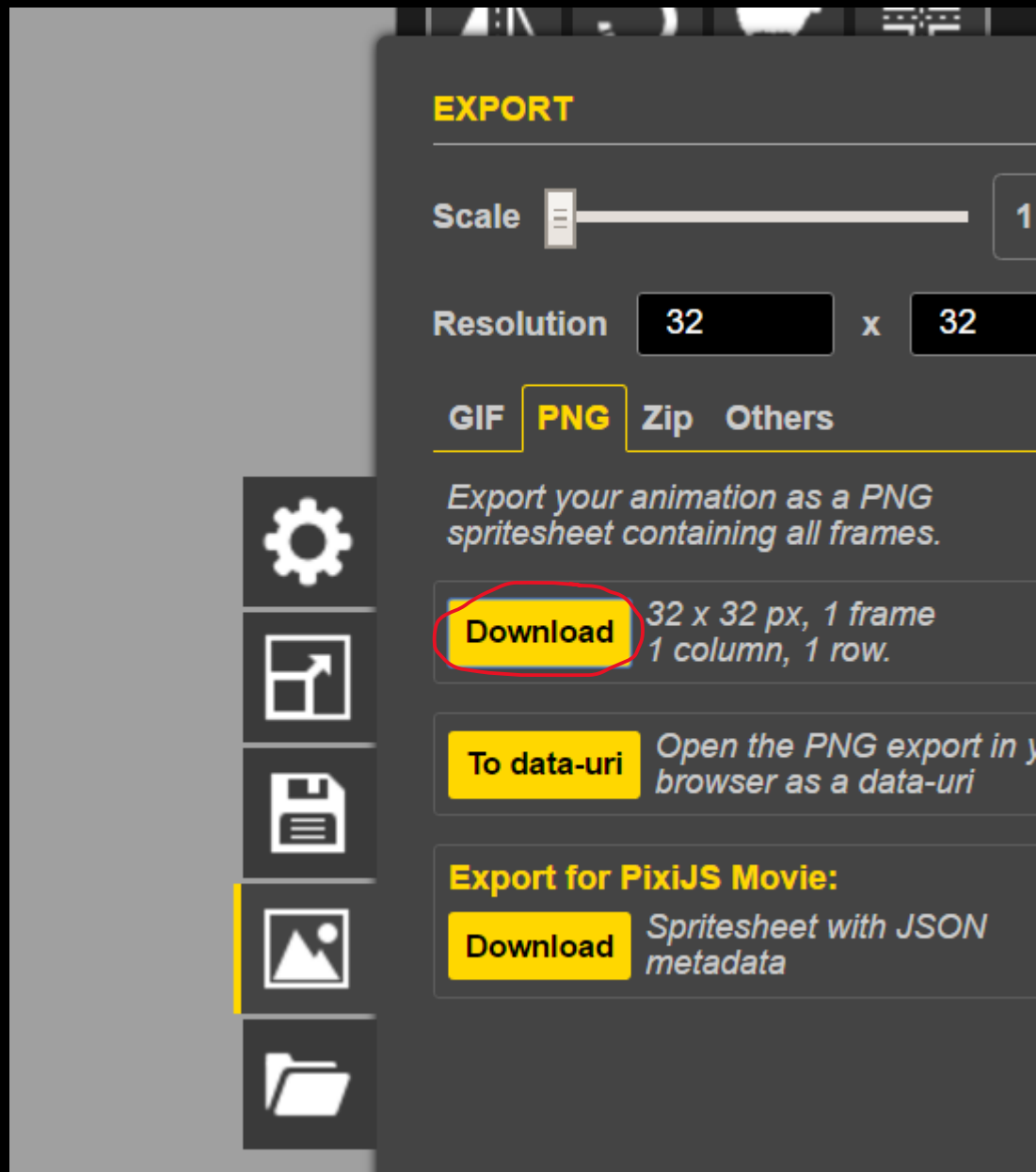
- ExtendViewport muss drei Dinge wissen:
  - Die minimale Anzahl an Welteinheiten in x- und y-Richtung
  - Die Kamera, um dieser die Informationen mitzuteilen
  - Wie groß das Fenster zu jedem Zeitpunkt ist

```
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

Repräsentiert Informationen wie Textur, Farbe, Position, Größe und Drehung eines Objekts der Spielwelt.

Kann verwendet werden, um festzulegen, wie ein Objekt auf dem Bildschirm dargestellt werden soll.

# 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.

# Eine Variable für jede Textur?

```
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);
    }
}
```

# enum für Objekttypen verwenden

```
public enum ObjektTyp {
    LIBGDX;

    private final String fileName;

    ObjektTyp() {
        this.fileName = name().toLowerCase(Locale.ROOT) + ".png";
    }

    public String getFileName() {
        return fileName;
    }
}
```

# Besser: Verwaltung durch AssetManager

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

private void maleMittig(ObjektTyp typ, Vector3 finger, float width) {
    String fileName = typ.getFileName();
    manager.load(fileName, Texture.class);
    manager.finishLoading();
    Texture texture = manager.get(fileName);
    float height = width * texture.getHeight() / texture.getWidth();
    batch.draw(texture, finger.x - width/2f, finger.y - height/2f, 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

```java
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.