

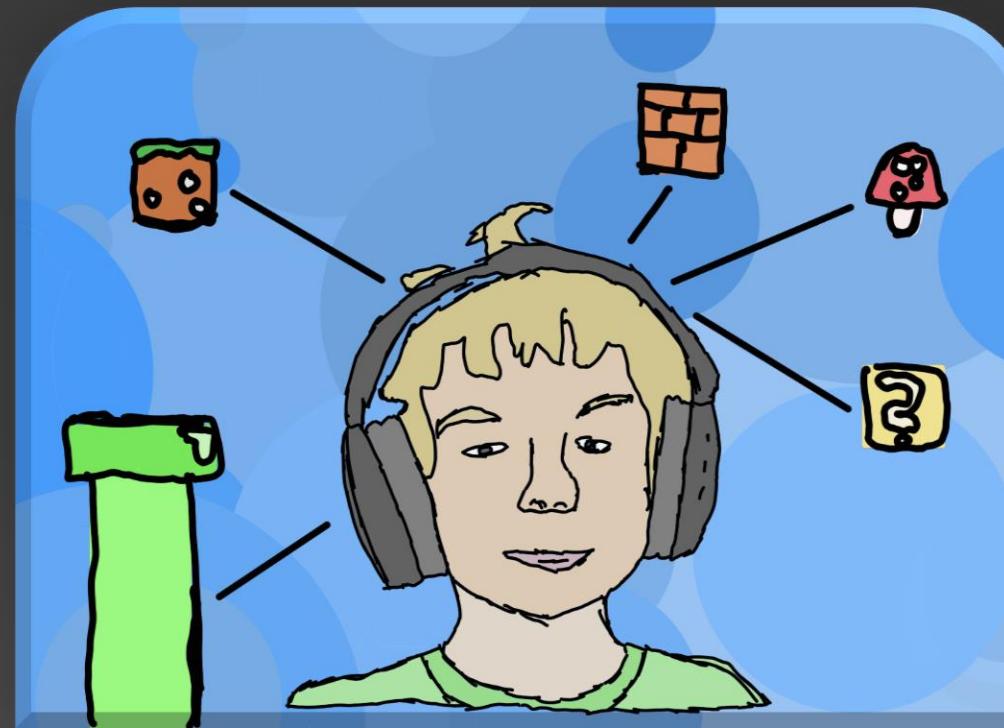
Spieleentwicklung mit Godot

Godot for Kids



Ich bin Olli...

- ❖ Hobbyspielleentwickler
- ❖ 3 Jahre Erfahrung mit Godot



Teile eines Spiels

- ❖ Art



- ❖ Sound

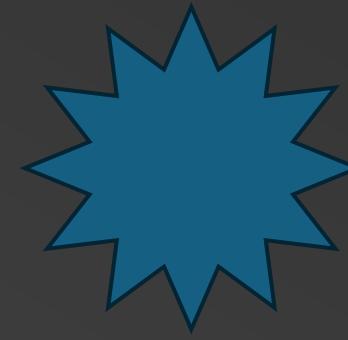
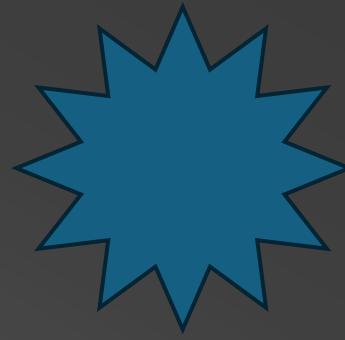
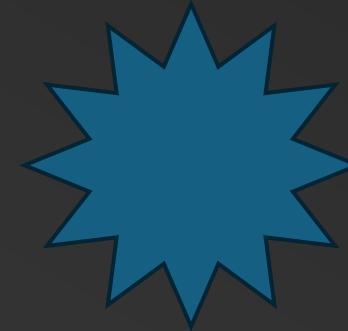
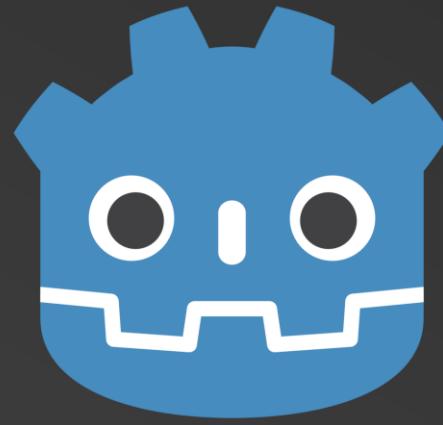
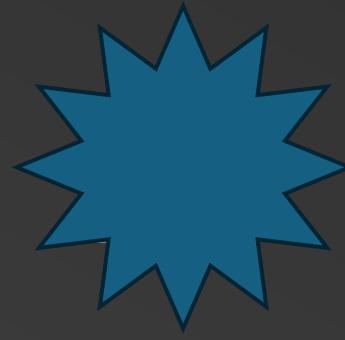


- ❖ Programmierung

1010
1010

- ❖ Gamedesign







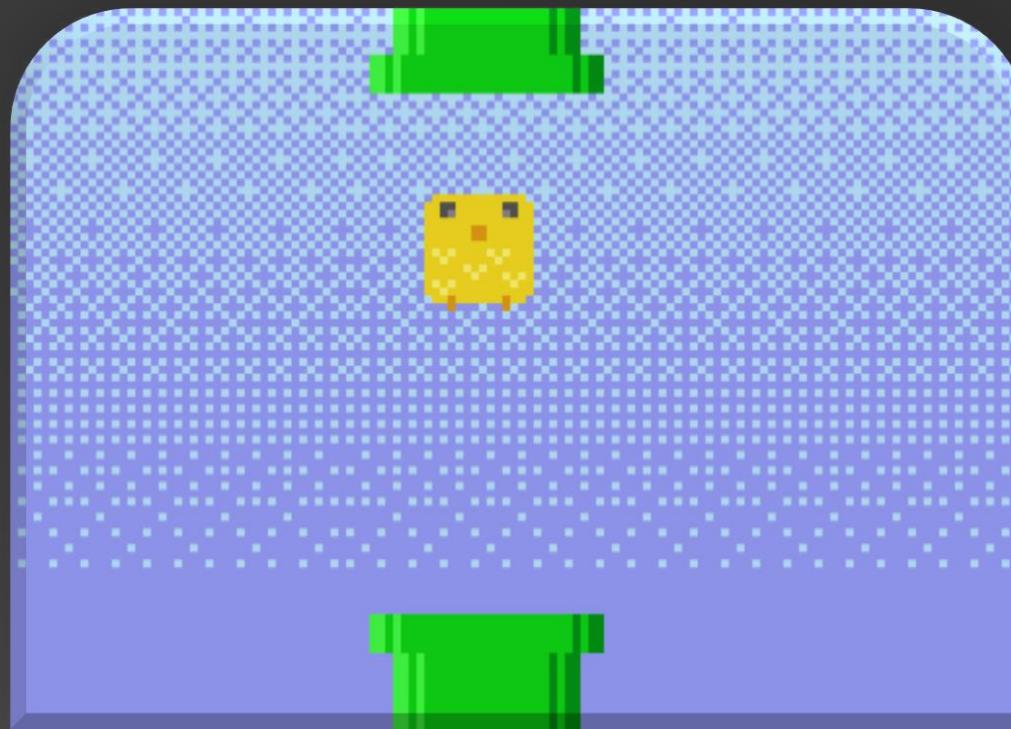
Warum Godot Engine?

- ❖ Es ist kostenlos und jeder, der Ahnung hat, kann Godot erweitern
- ❖ Mit Godot kann man sich sehr gut eigene Tools bauen
(Was das heißt, seht ihr später)
- ❖ Es ist einfach und intuitiv



Einsteiger Projekt: Tappy Bird

- ❖ Vogel fliegt zwischen Röhren
- ❖ Die Röhren dürfen nicht berührt werden

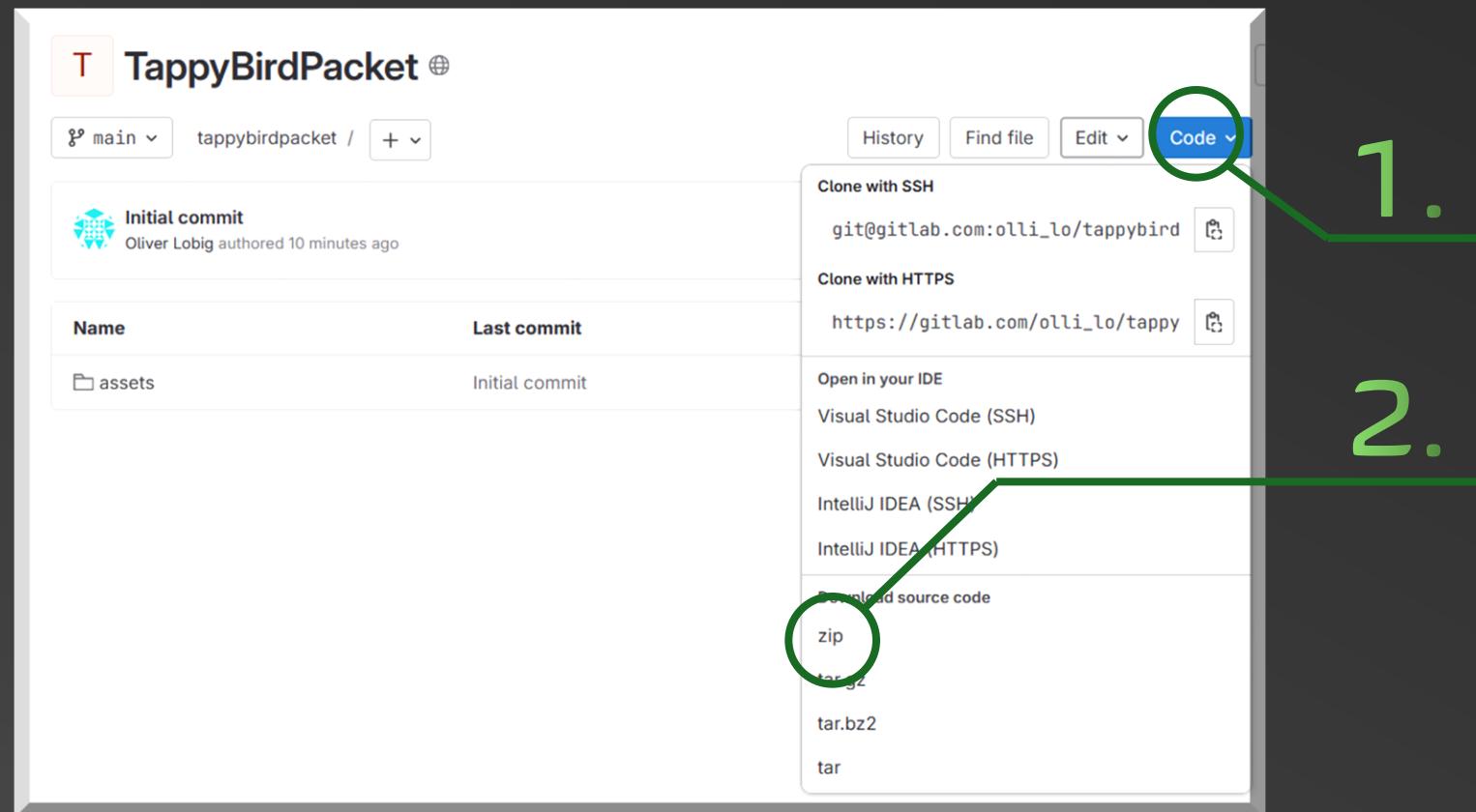


Vorbereitung: Spielpaket herunterladen

- ❖ Öffnen: t.ly/kp7lX



Paket herunterladen

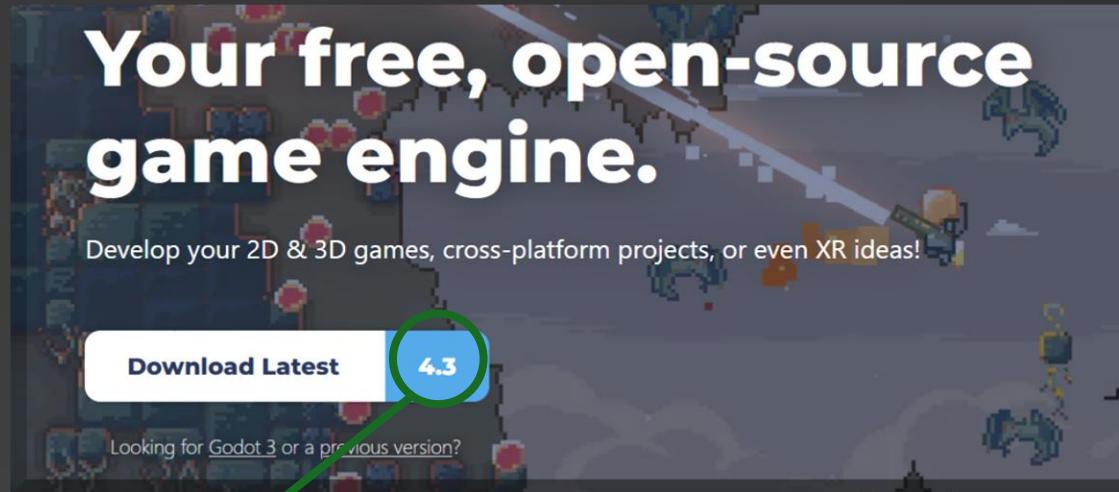


Vorbereitung: Godot herunterladen

- ❖ Öffnen: <https://t.ly/oqNTg>



Godot herunterladen



1.

2.



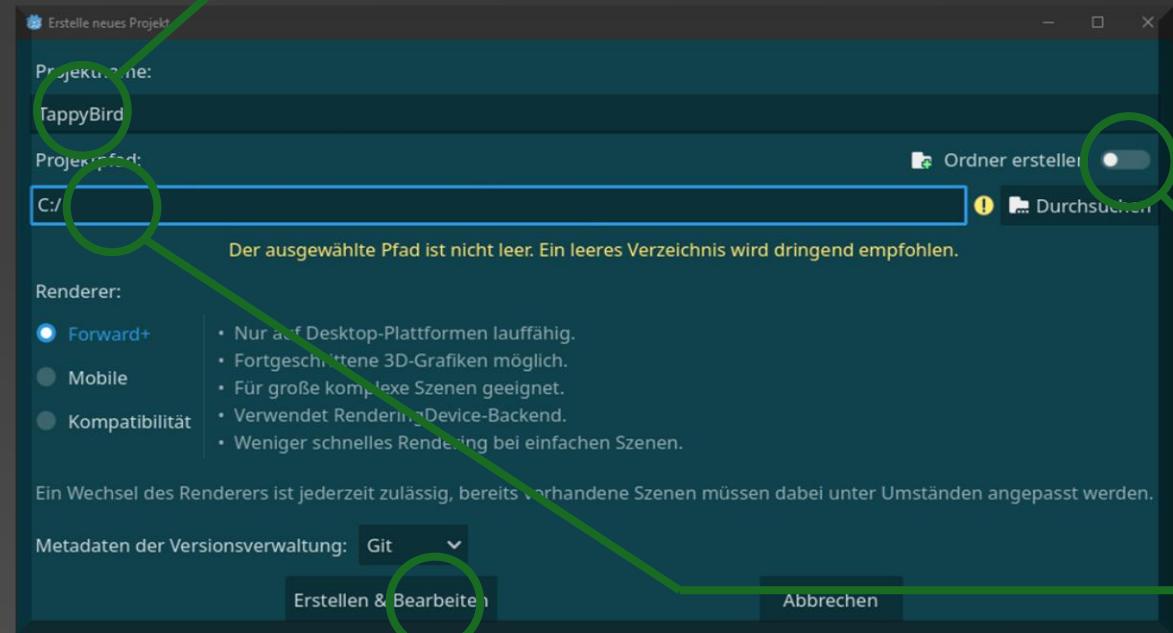
Umräumen & Los geht's

- ❖ Godot Ordner erstellen
- ❖ TappyBird Paket in den Ordner ziehen
- ❖ TappyBird Paket entpacken
- ❖ Paket umbenennen in „TappyBird“
- ❖ Godot-Download in den Ordner ziehen
- ❖ .zip extrahieren
- ❖ ZIP-Dateien löschen
- ❖ Godot ausführen

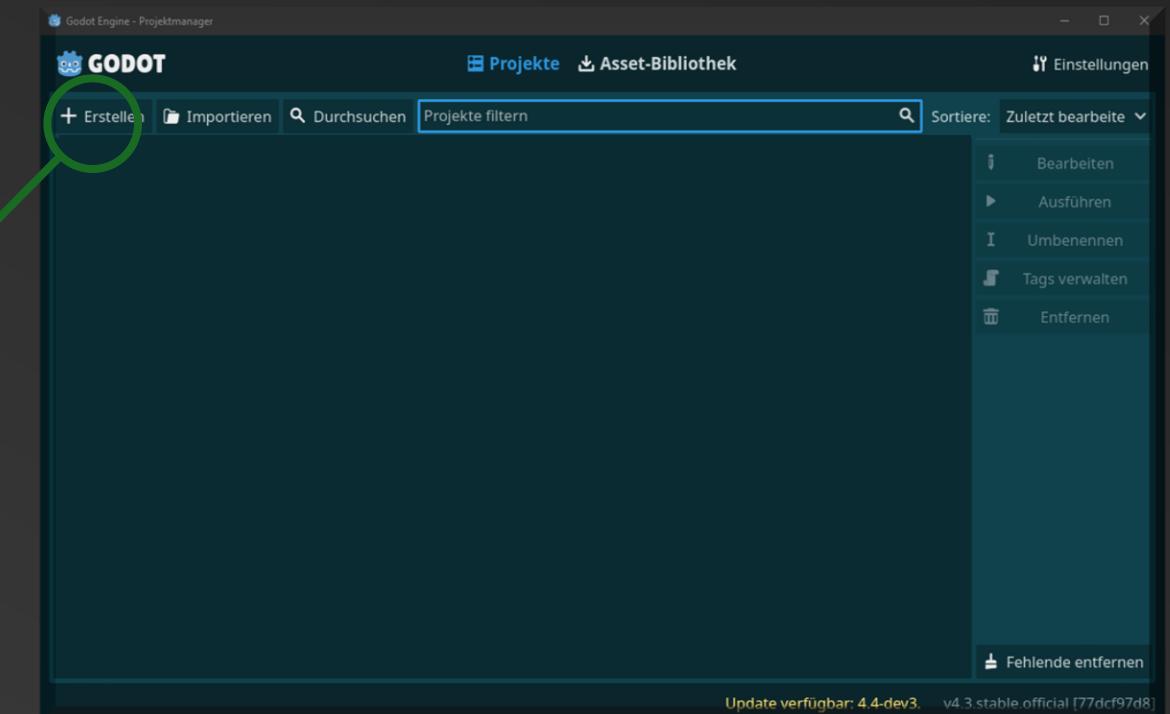


Projekt erstellen

2. *TappyBird* eingeben



1.

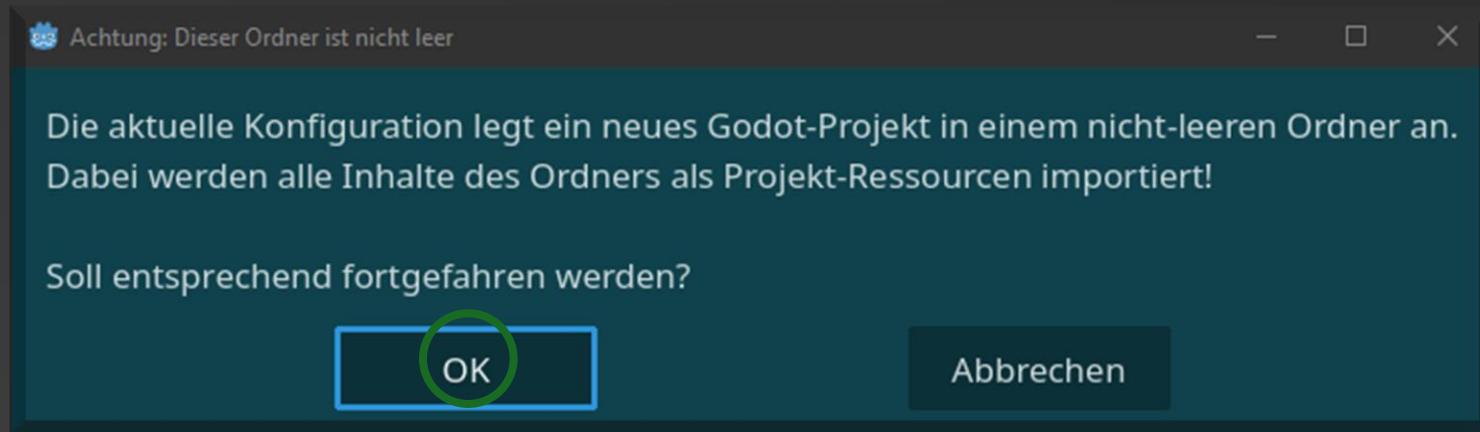


3. Ordner erstellen *aus*

4. Den Pfad zum TappyBird-Ordner angeben

5. Projekt erstellen

Ja, aber...



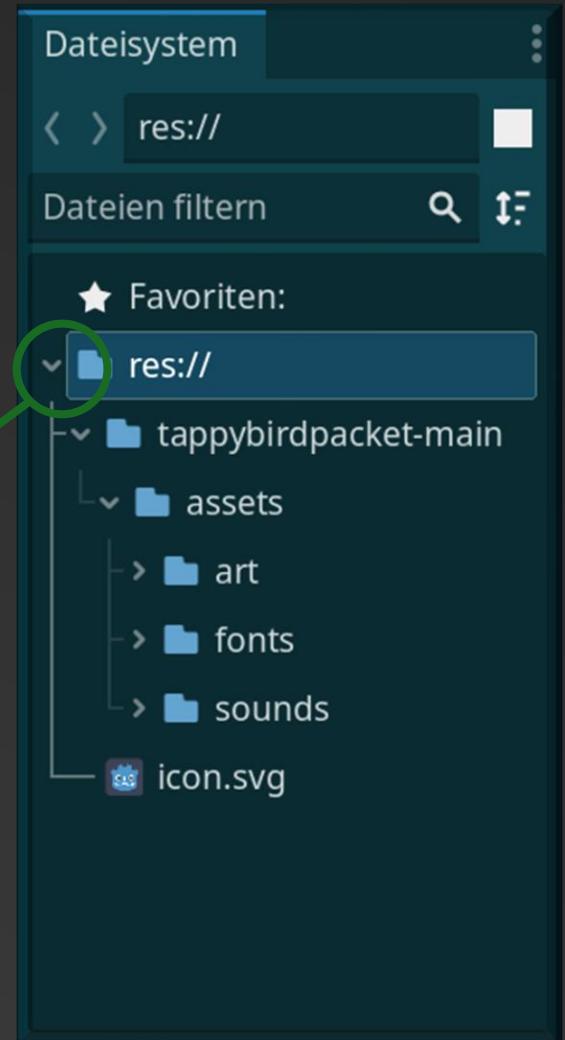
- ❖ In unserem Projekt soll das TappyBird-Paket schon enthalten sein
- ❖ Sonst Projekte *immer* in einem *leeren* Ordner erstellen (Ordner-erstellen Option an lassen)

Das Tappy-Bird Paket

- ❖ In dem Paket habe ich Grafik und Sound vorbereitet



Eventuell ausklappen



Godot-Spiele bestehen aus Szenen

- ❖ Szenen sind so wie Theaterszenen
- ❖ Eine Szene kann andere Szenen enthalten

Szenen bestehen aus Nodes

- ❖ Nodes sind kleine Bauteile von Spielen
- ❖ Sie werden in einer Baumstruktur angeordnet

Beispiel für Tappy-Bird:



Hauptsache Hauptszene

- ❖ Wir machen ein 2D-Spiel



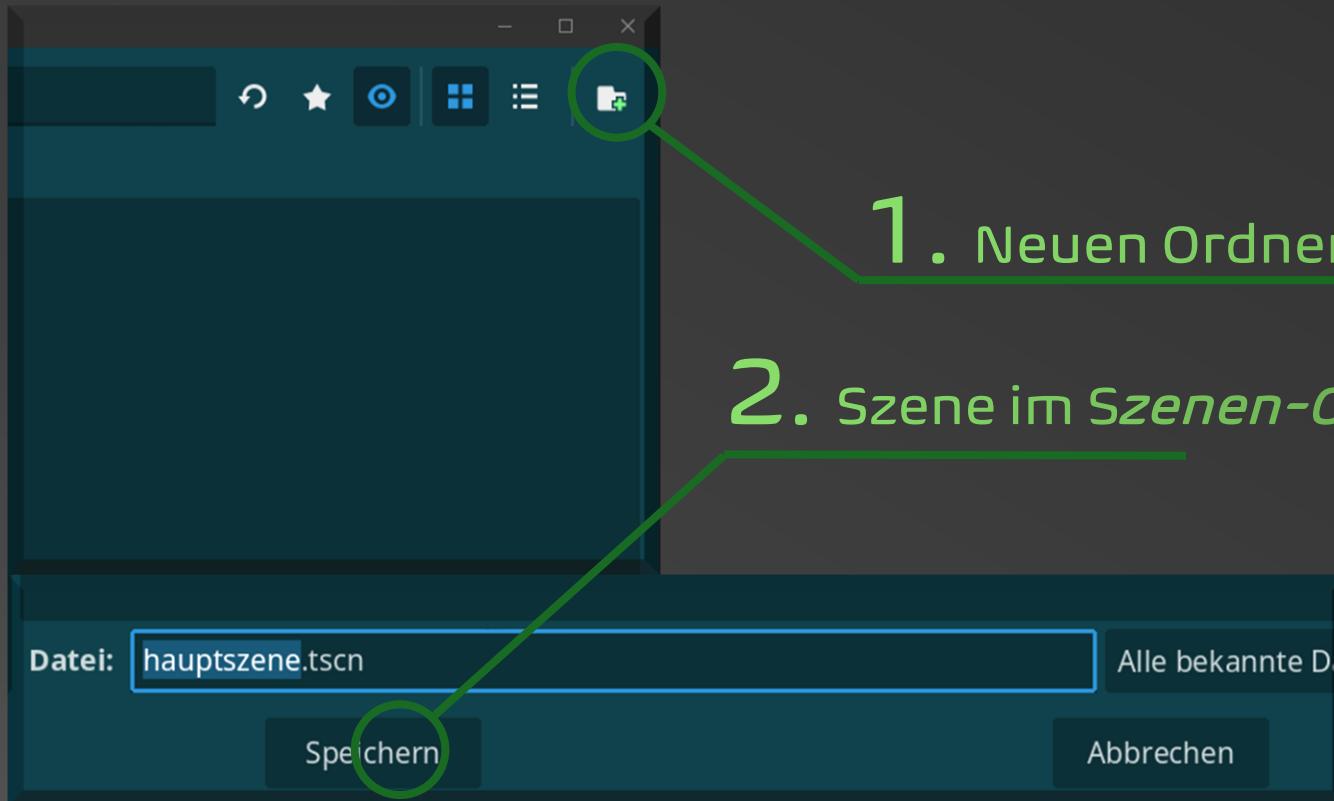
Doppelklicken und in *Hauptszene* umbenennen

2D-Szene erstellen



Geschafft! Erstmal speichern

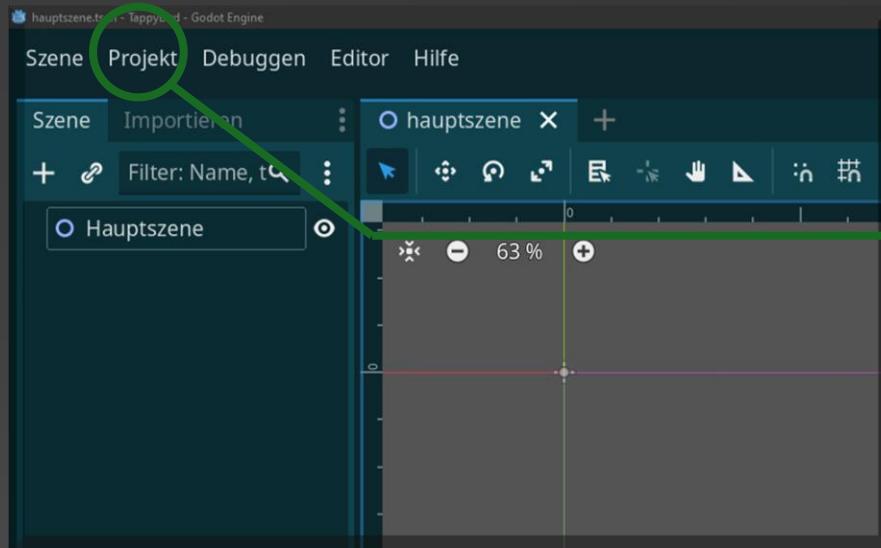
- ❖ Strg + S drücken



1. Neuen Ordner erstellen und *Szenen* nennen

2. Szene im *Szenen-Ordner* speichern

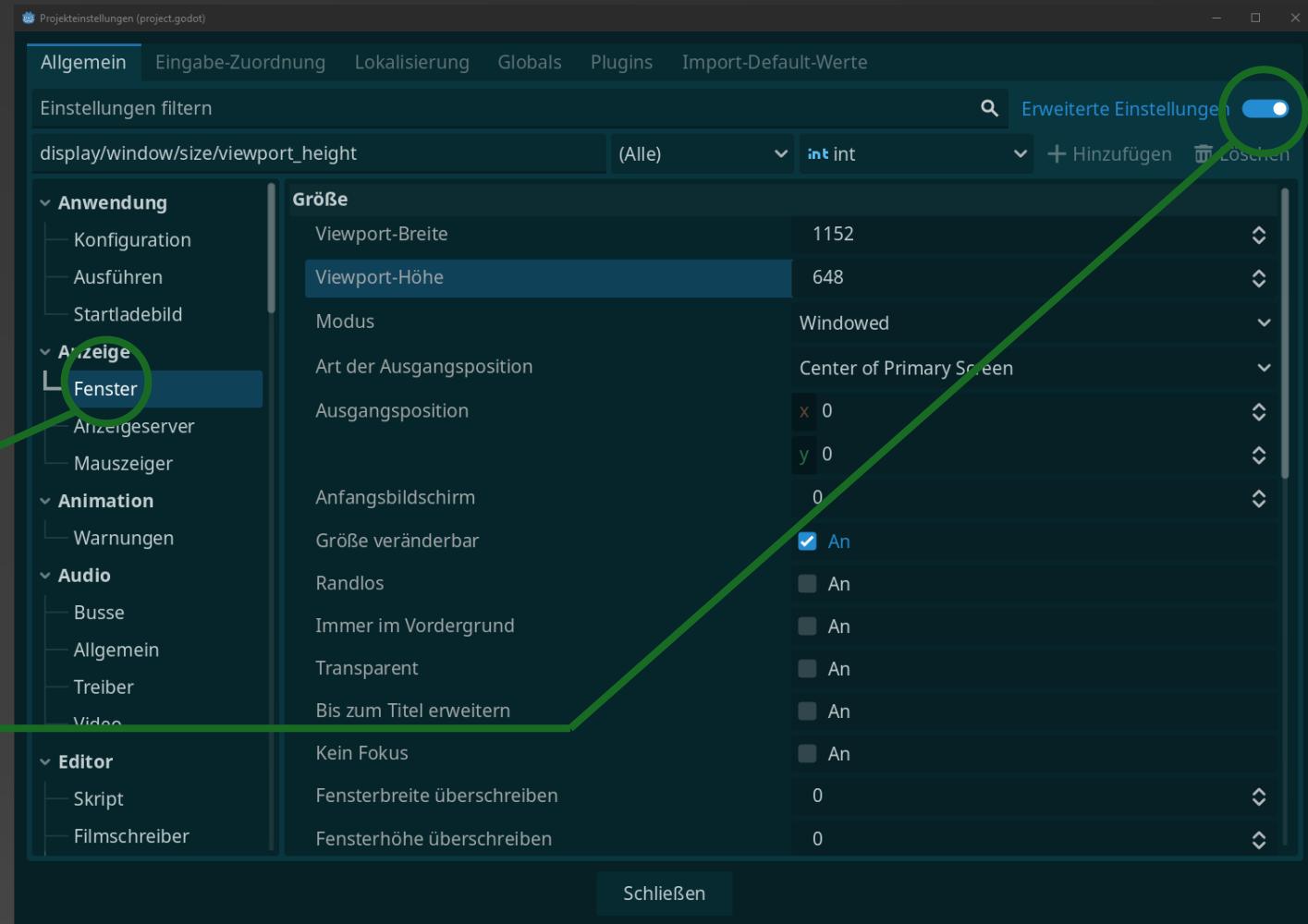
Spiel-Einstellungen



1. Projekteinstellungen öffnen

Fenster-Einstellungen

1. Fenster anklicken
2. Erweiterte Einstellungen aktivieren



Einstellen!

Auf die Werte im Bild
einstellen

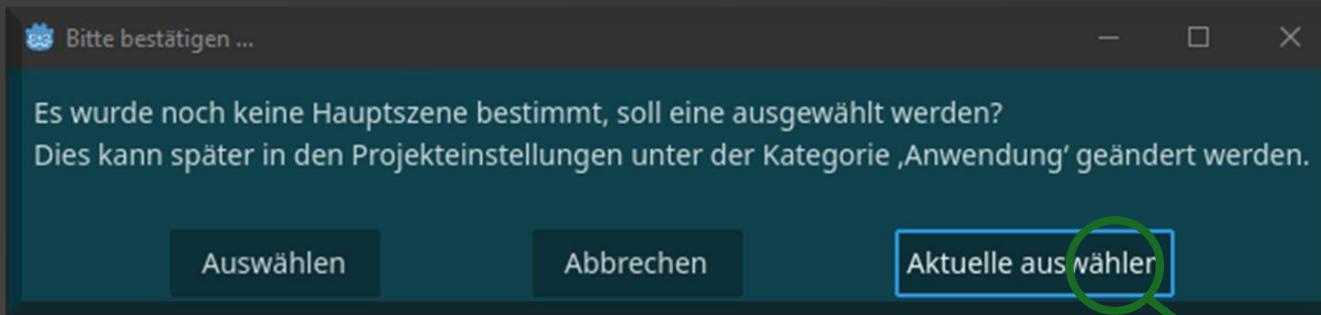
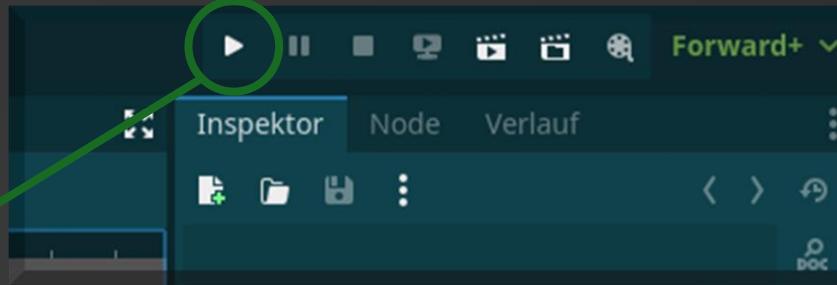
The screenshot shows a configuration interface with the following settings:

- Größe (Size):**
 - Viewport Breite: 135
 - Viewport Höhe: 240
 - Modus: Windowed
 - Alt der Ausgangsposition
 - Ausgangsposition
 - Anfangsbildschirm
 - Größe veränderbar
 - Randlos
 - Immer im Vordergrund
 - Transparent
 - Bis zum Titel erweitern** (Selected)
 - Kein Fokus
 - Fensterbreite überschreiben
 - Fensterhöhe überschreiben
- Energiesparen (Energy Save):**
 - Bildschirm eingeschaltet lassen
 - Bildschirm eingeschaltet lassen.editor_hint
- Unterfenster (Subwindows):**
 - Unterfenster einbetten
- Streckung (Scaling):**
 - Modus: viewport
 - Seitenverhältnis
 - Skalierung
 - Skalierungsmodus: keep
- DPI:**
 - 1 fractional

Schließen (Close) button is at the bottom right.

Spielen!

1. Spiel starten



2. Aktuelle Szene als Start-Szene auswählen

Tada

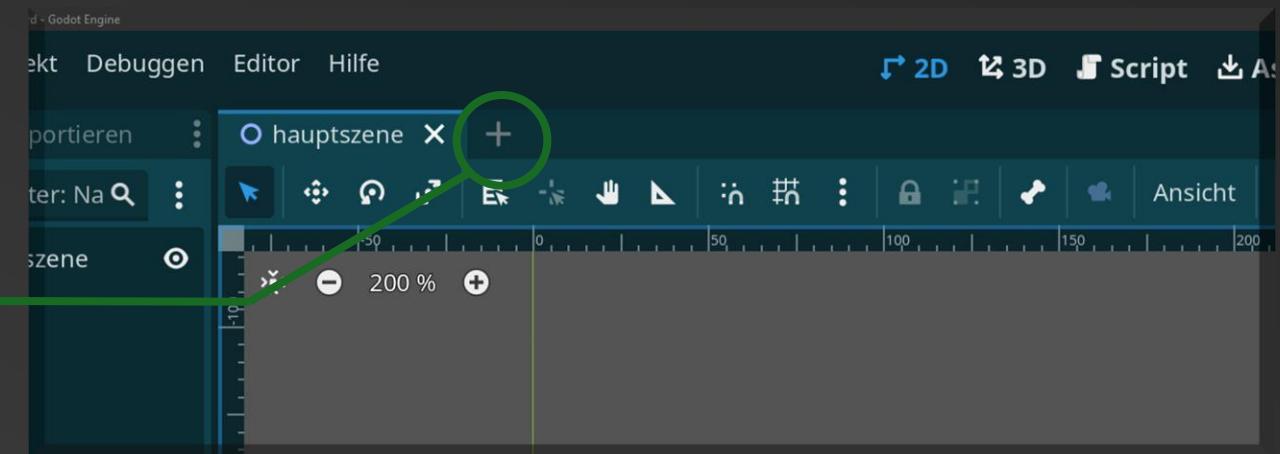
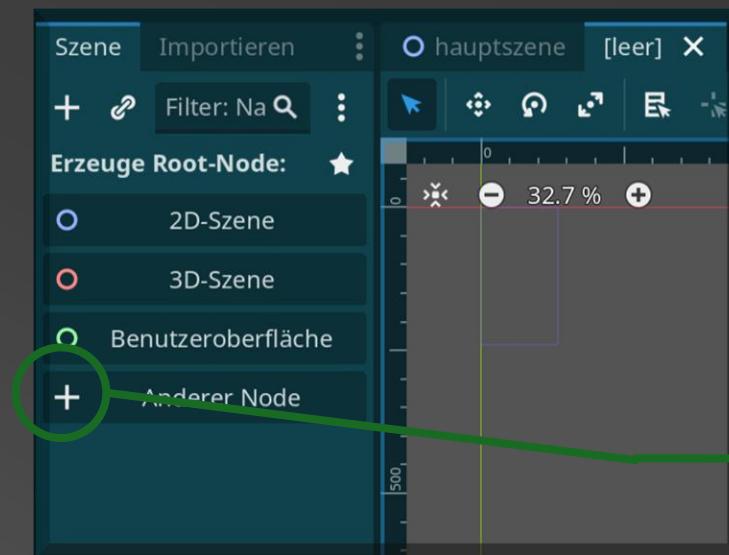
- ❖ Alles ist vorbereitet
- ❖ Wir können anfangen das Spiel zu entwickeln



Der Vogel



1. Neue Szene erstellen



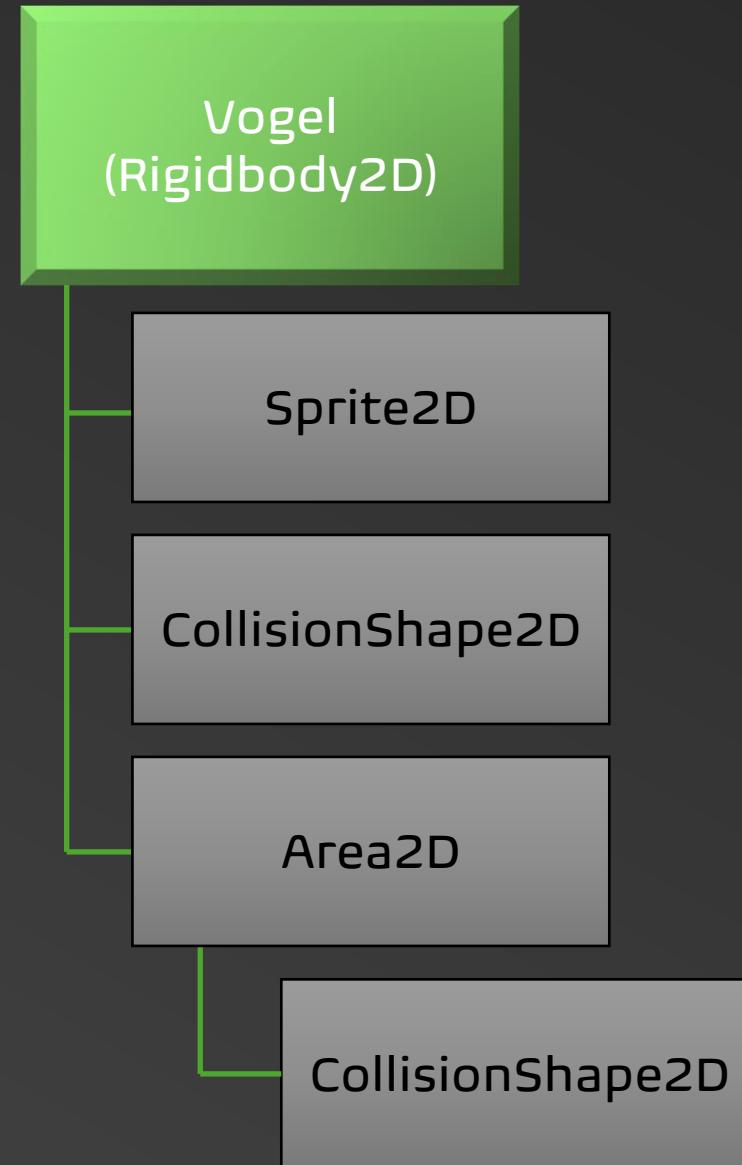
2. *RigidBody2D* als Hauptnode auswählen

3. *RigidBody2D* in *Vogel* umbenennen

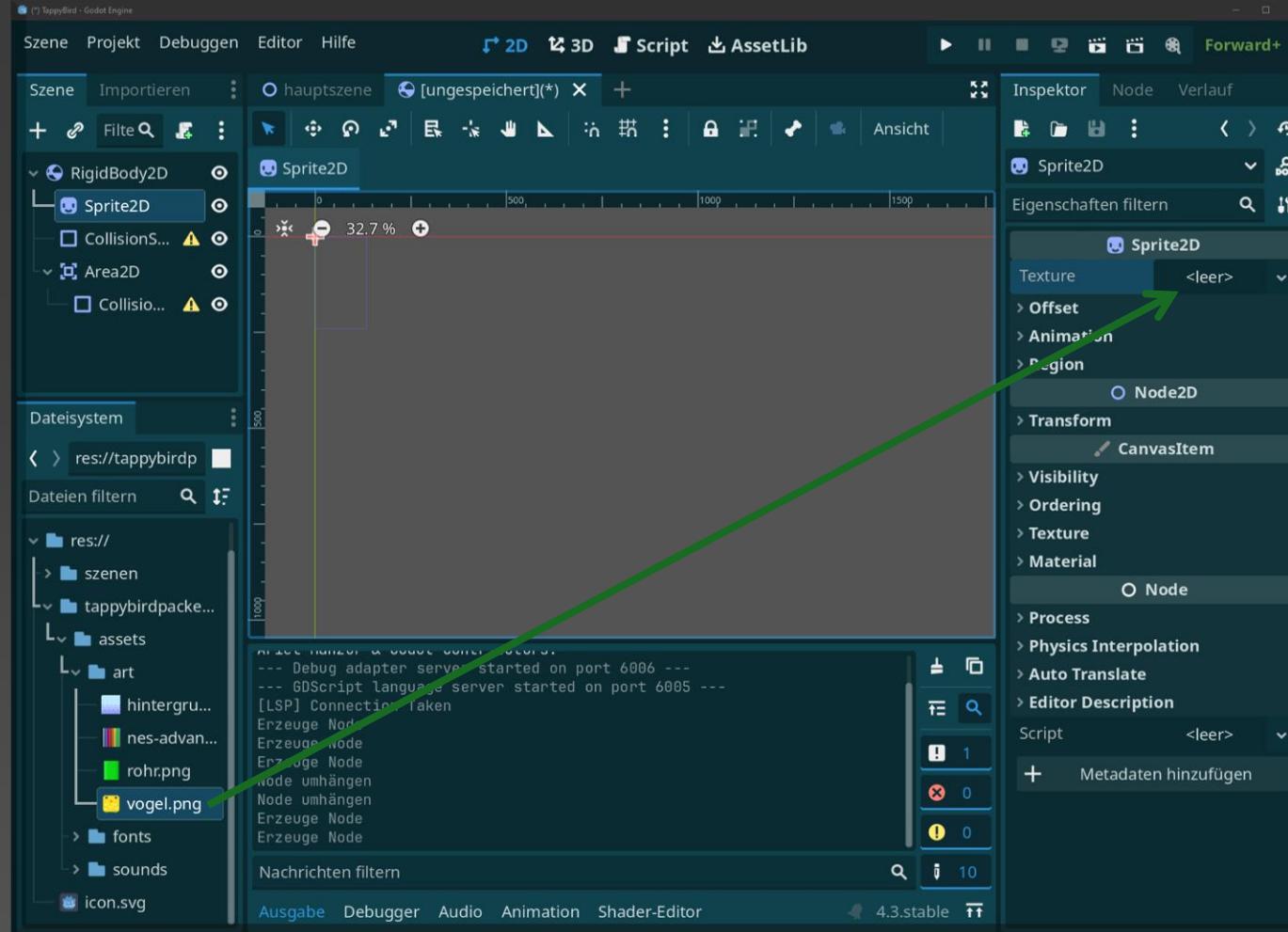
Der Vogel



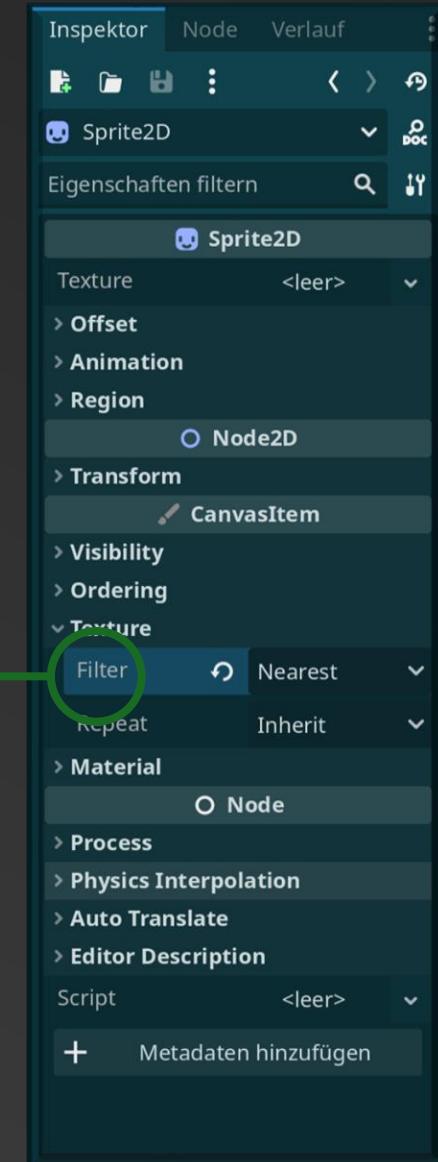
- ❖ Unternodes hinzufügen →
- ❖ Speichern (Strg + S) im Szenen-Ordner



Der Vogel - Aussehen



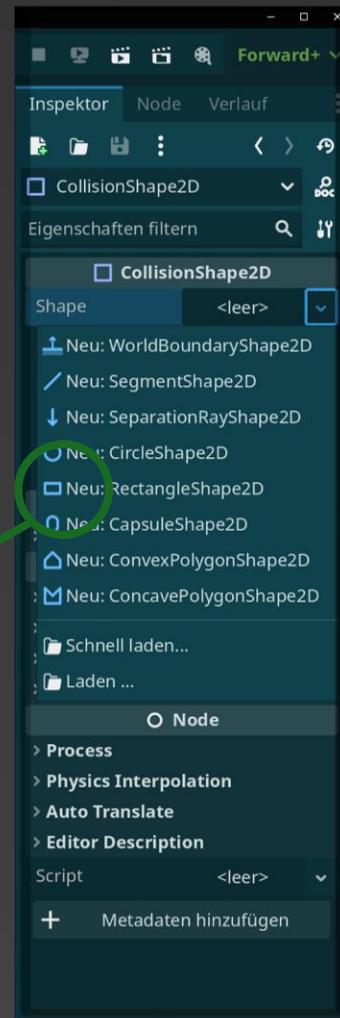
2. Filter unter
Texture auf
Nearest setzen:
Vogel ist nicht
mehr
verschwommen



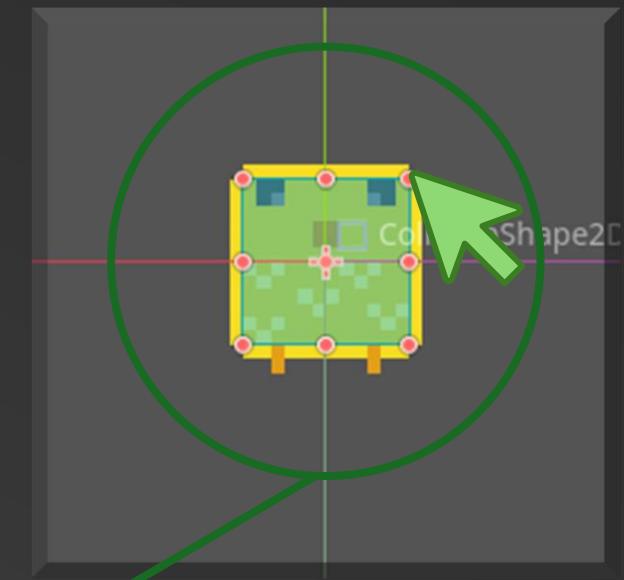
Der Vogel - Kollision



1. In der CollisionShape eine
neue RectangleShape2D
erstellen

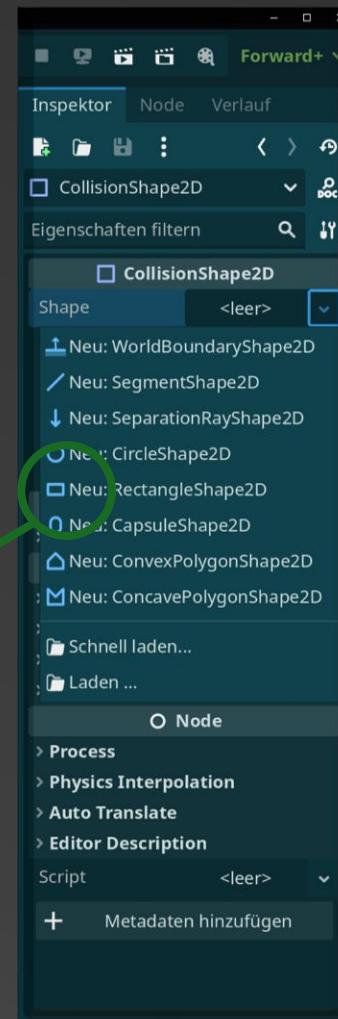


2. Größe auf Vogel
einstellen (1px kleiner)

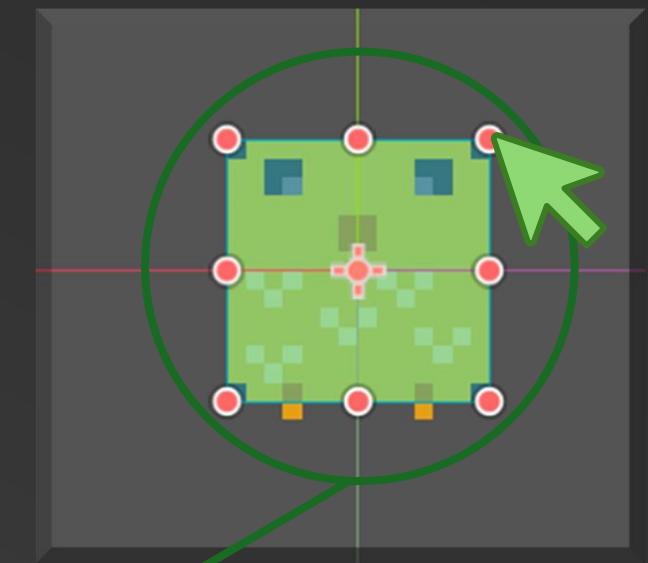


Der Vogel - Detektion

1. In der zweiten
CollisionShape eine neue
RectangleShape2D erstellen



2. Größe auf Vogel
einstellen



Der Vogel - Gewicht



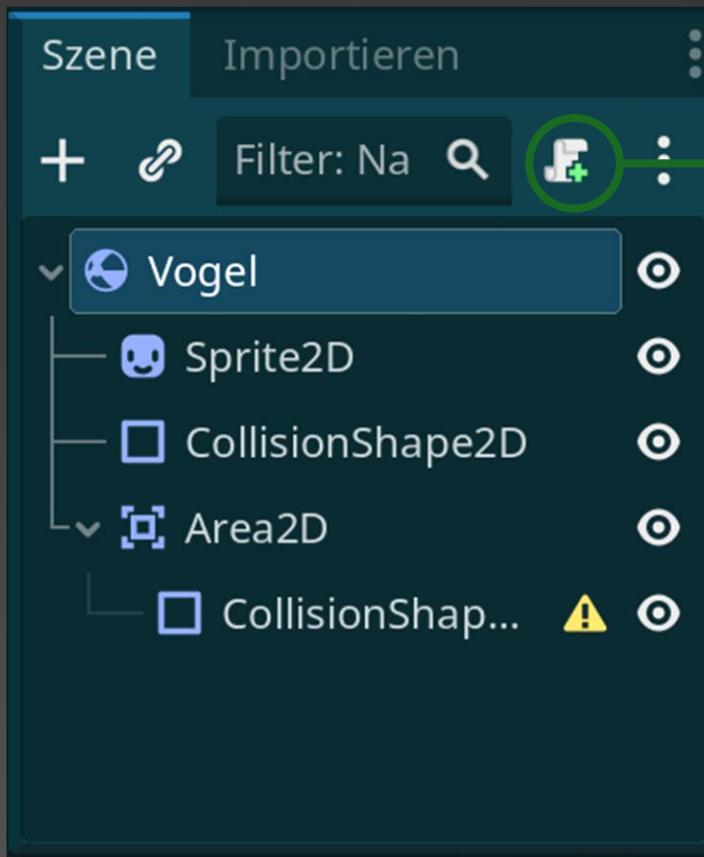
The image shows the Unity Editor interface. On the left, the Hierarchy window is open, showing a scene with a single object named "Vogel". This object contains a "Sprite2D" component and an "Area2D" component, which in turn contains a "CollisionShape2D" component. A green arrow points from the "Vogel" object in the Hierarchy to the "Rigidbody2D" component in the Inspector window on the right. The Inspector window displays the following settings for the "Rigidbody2D" component:

- Mass: 0.1 kg (highlighted with a green circle)
- Physics Material Override: <leer>
- Gravity Scale: 1
- Mass Distribution (button)
- Deactivation (button)
- Solver (button)
- Linear (button)
- Angular (button)
- Constant Forces (button)

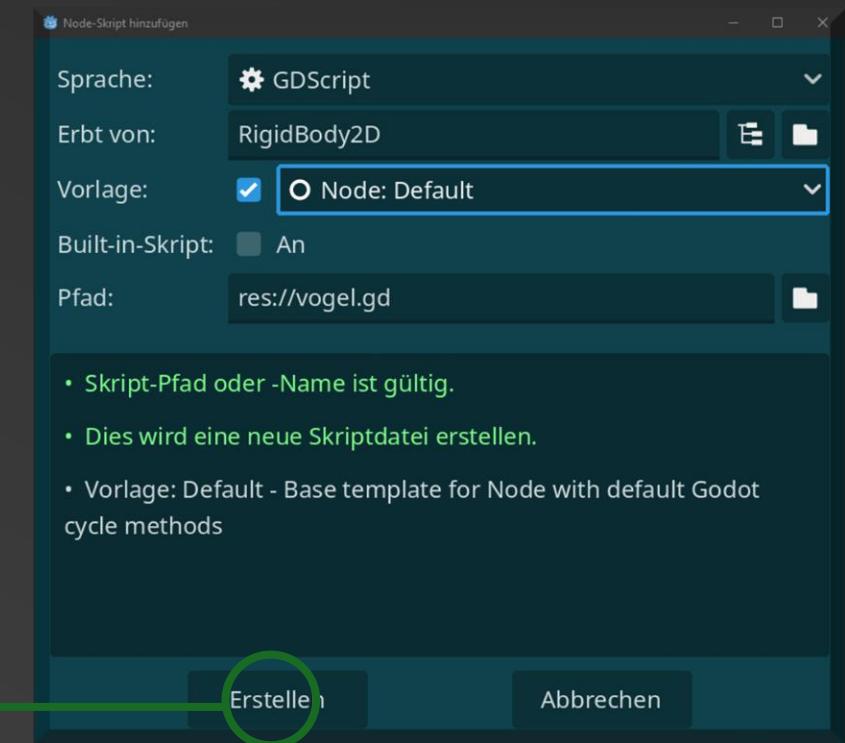
❖ Der Vogel ist noch zu schwer

Stelle das Gewicht des Vogels auf *0.1kg*

Der Vogel - Springen



1. Dem *Hauptnode* ein neues Skript hinzufügen



2. Auf *Erstellen* klicken

Der Vogel - Springen



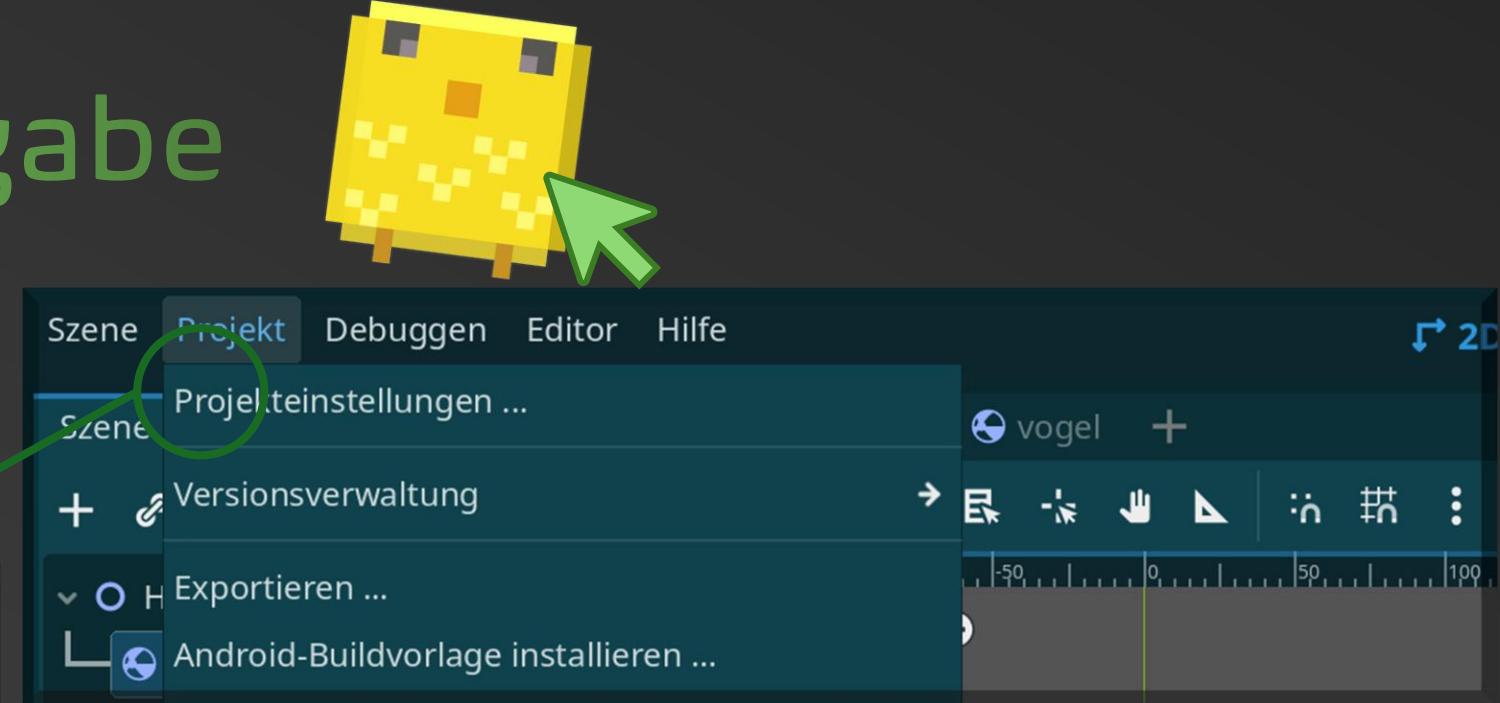
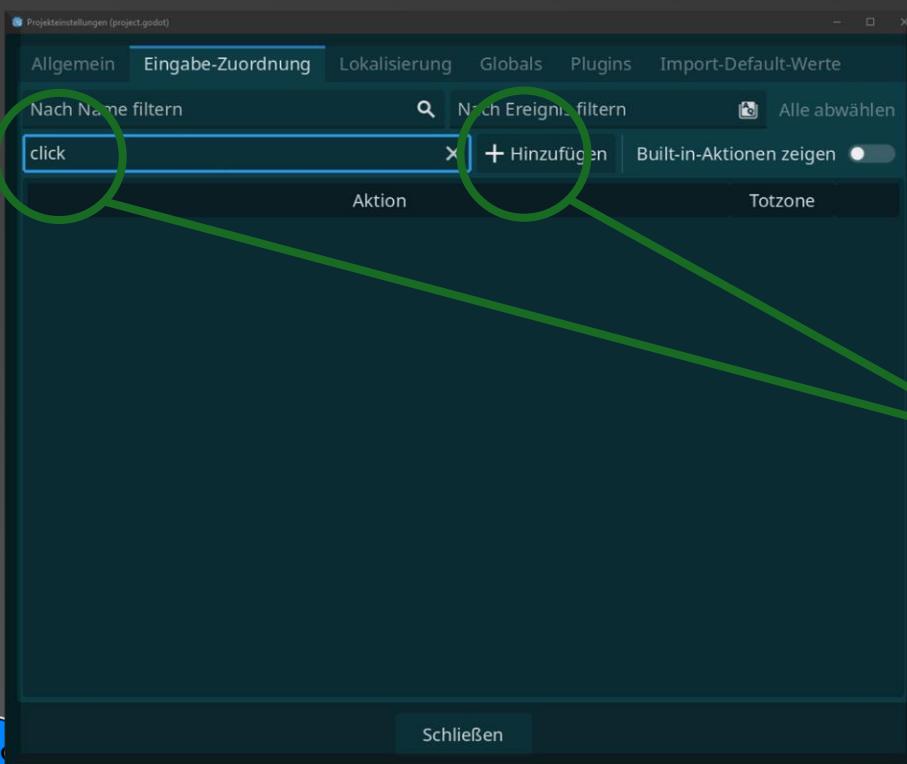
- ❖ Alles nach einem **#** nicht mit abschreiben → es hat keine Funktion, sondern dient nur zur Erklärung des Codes

```
Suchen Springe zu Debuggen Online-Dokumentation
1  extends RigidBody2D
2
3  @export var JUMP_HEIGHT: float = 50.0 #Sprunghöhe
4  const UP_VECTOR: Vector2 = Vector2.UP #Richtung, die nach Oben zeigt
5
6
7  func _input(event: InputEvent) -> void: #Wird bei einer Eingabe aufgerufen
8    if event.is_action_pressed("click"): #Wenn geclikkt wird
9      apply_impulse(UP_VECTOR * JUMP_HEIGHT) #Vogel nach oben werfen
10
```

Der Vogel - Eingabe

1.

Projekteinstellungen
öffnen



2. In Eingabe-Zuordnung eine neue
click-Aktion hinzufügen

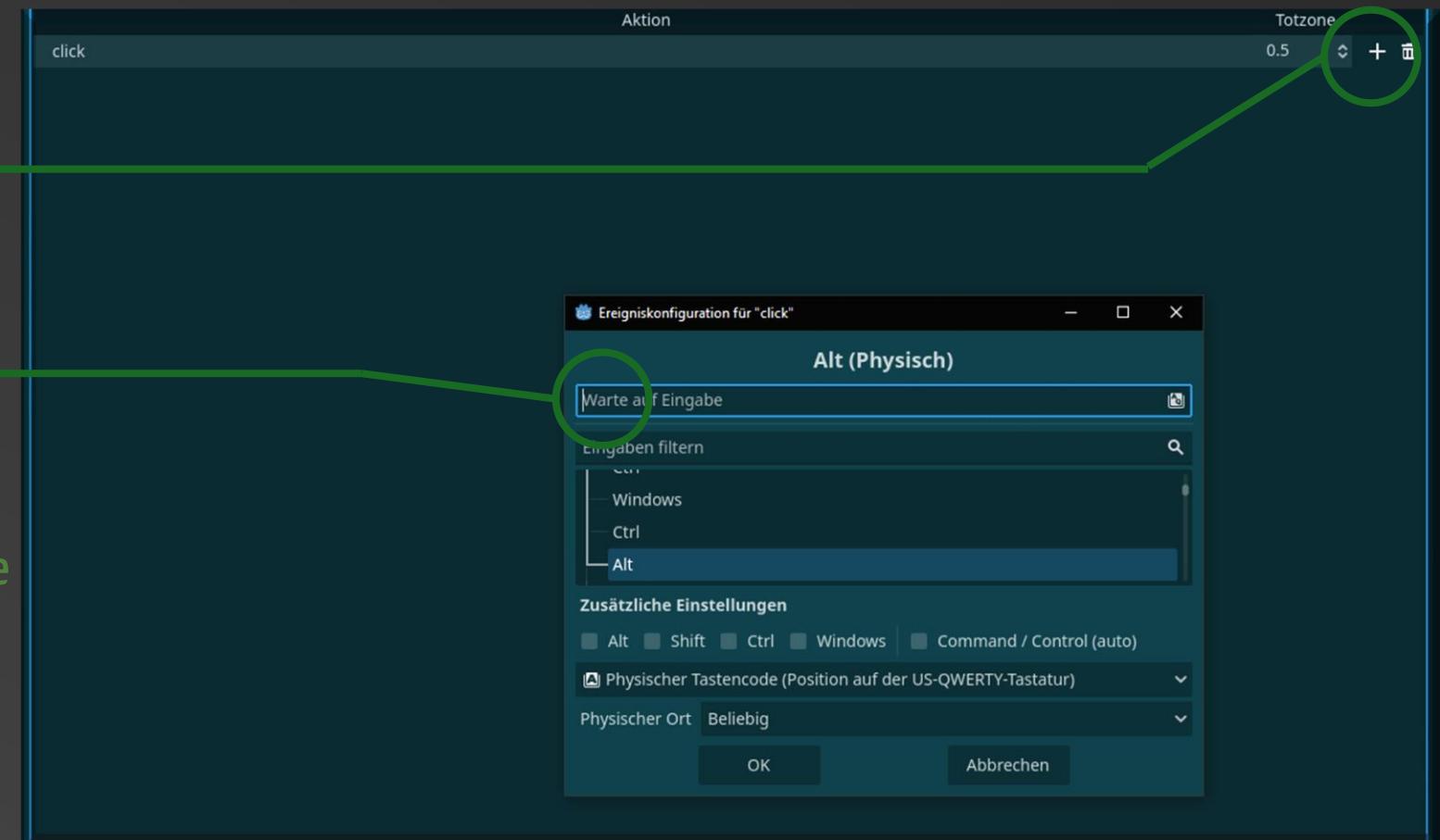
Der Vogel - Eingabe



1. Click Ereignissen
zuordnen

2. Linke Maustaste
drücken

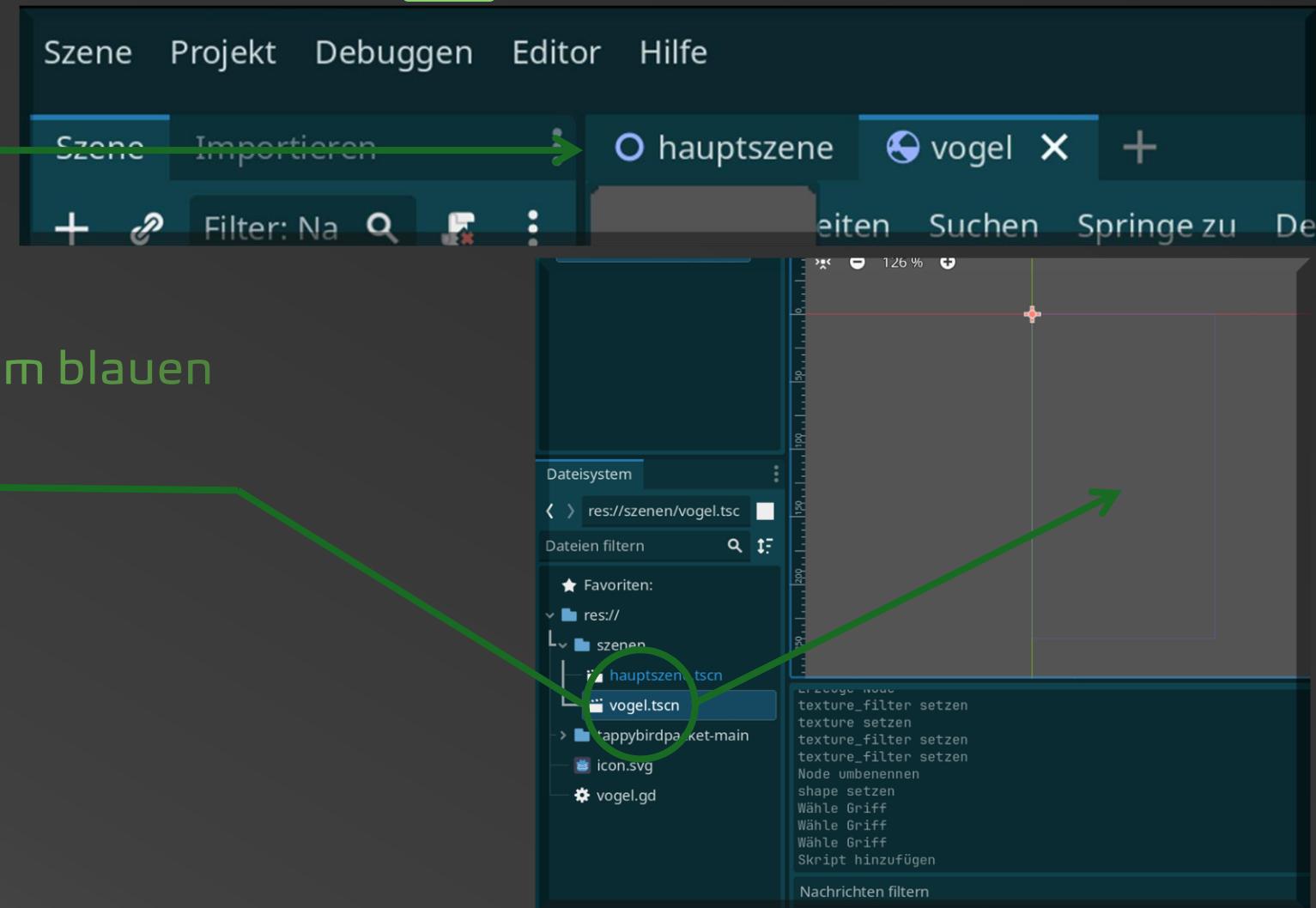
3. Für andere Eingabe
Methoden wiederholen



Der Vogel - Einfügen



- ❖ Skript speichern
- ❖ Hauptszene öffnen



3. Vogel in die *Mitte* vom blauen Rahmen ziehen

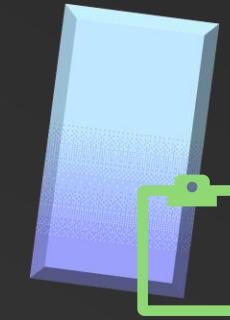
- ❖ Speichern
- ❖ Spiel starten

Der Vogel - Das haben wir gemacht



- ❖ Der Vogel kann fallen
- ❖ Wenn man auf den Bildschirm klickt, springt der Vogel

Hintergrund- Einfügen



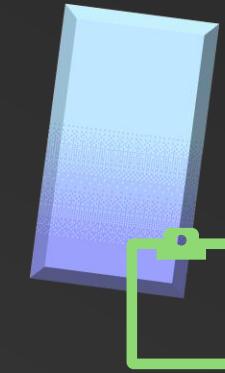
- ❖ Der Hintergrund soll sich bewegen, nicht der Vogel
- ❖ Damit der Hintergrund sich bewegen kann, muss er größer als ein Bildschirm sein
- ❖ Wir wollen den Hintergrund 4-mal einfügen

1. Neuen Node 2D unter der Hauptszene einfügen

2. Den Node 2D *Hintergrund* nennen

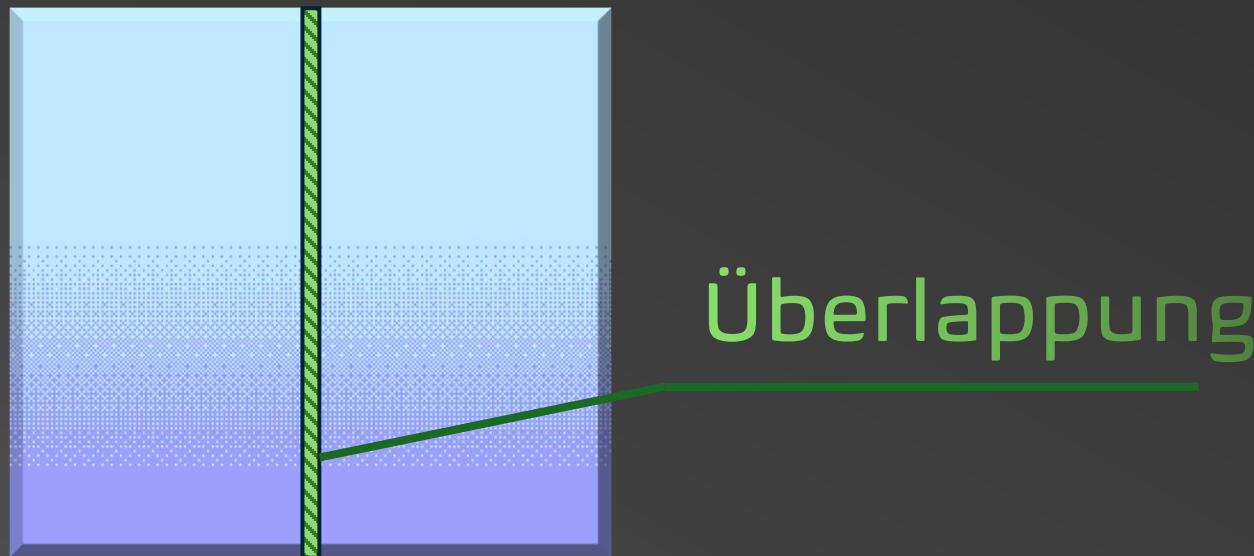


Hintergrund- Einfügen

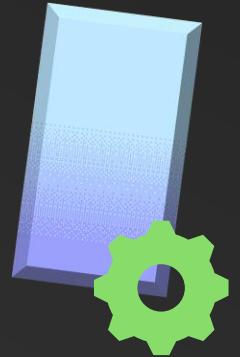


1. 4 Hintergründe hintereinander in das Spiel ziehen. Dabei muss der Hintergrund-Node ausgewählt sein

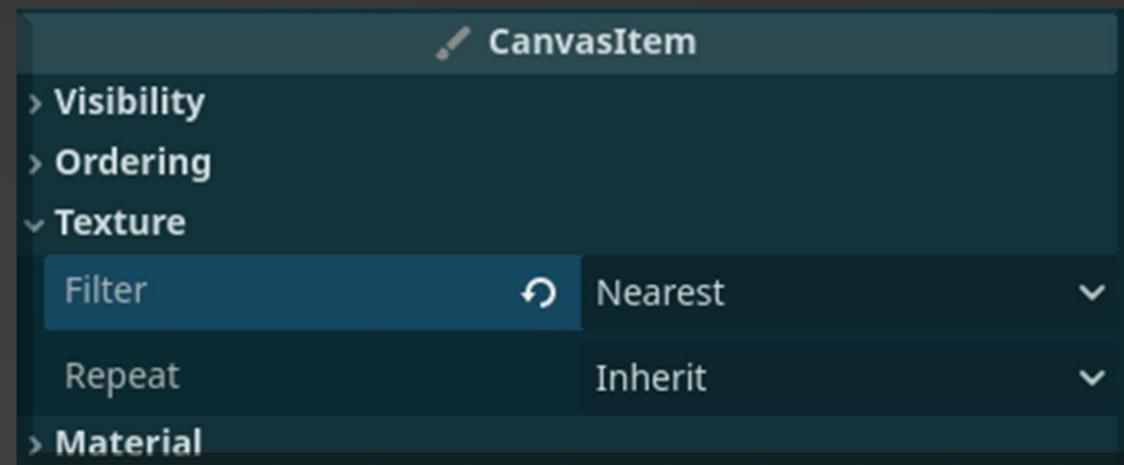
Tipp: Bei dieser Hintergrund-Grafik sieht es besser aus, wenn sich jeder Hintergrund *einen Pixel* überlappt!



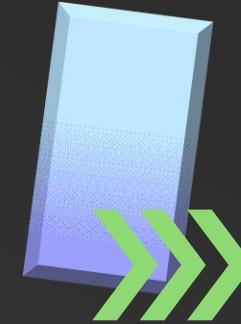
Hintergrund-Textur-Einstellungen



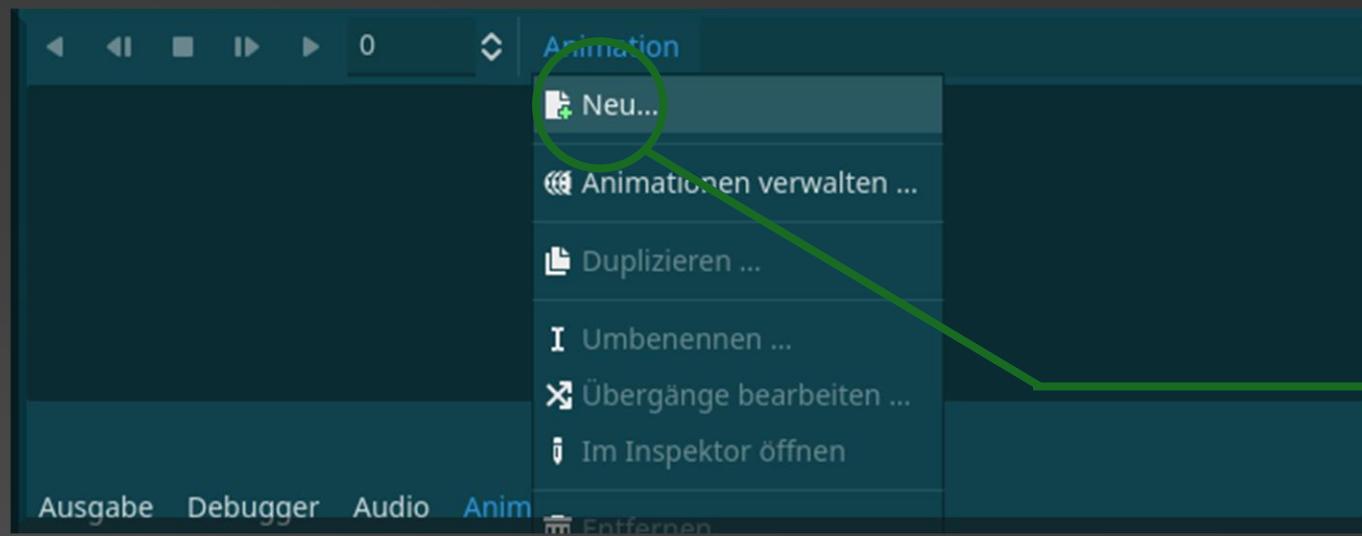
Setze bei dem Hintergrund-Hauptnode unter Texture den Filter auf Nearest
→ Der Hintergrund ist nicht mehr verschwommen



Hintergrund- Animation

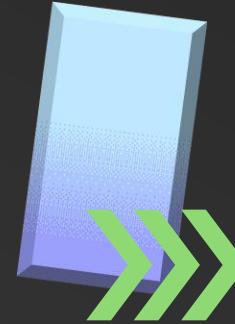


1. Ein *Animation-Player* unter dem Hintergrund-Node einfügen

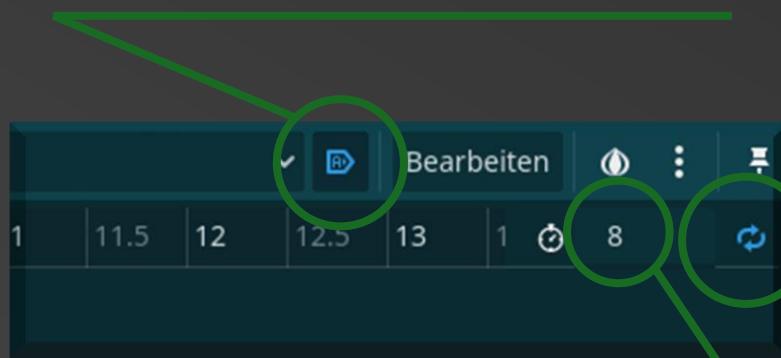


2. Eine neue Animation mit dem Namen *Loop* hinzufügen

Hintergrund- Animation



1. *Autostart* bei der Animation aktivieren



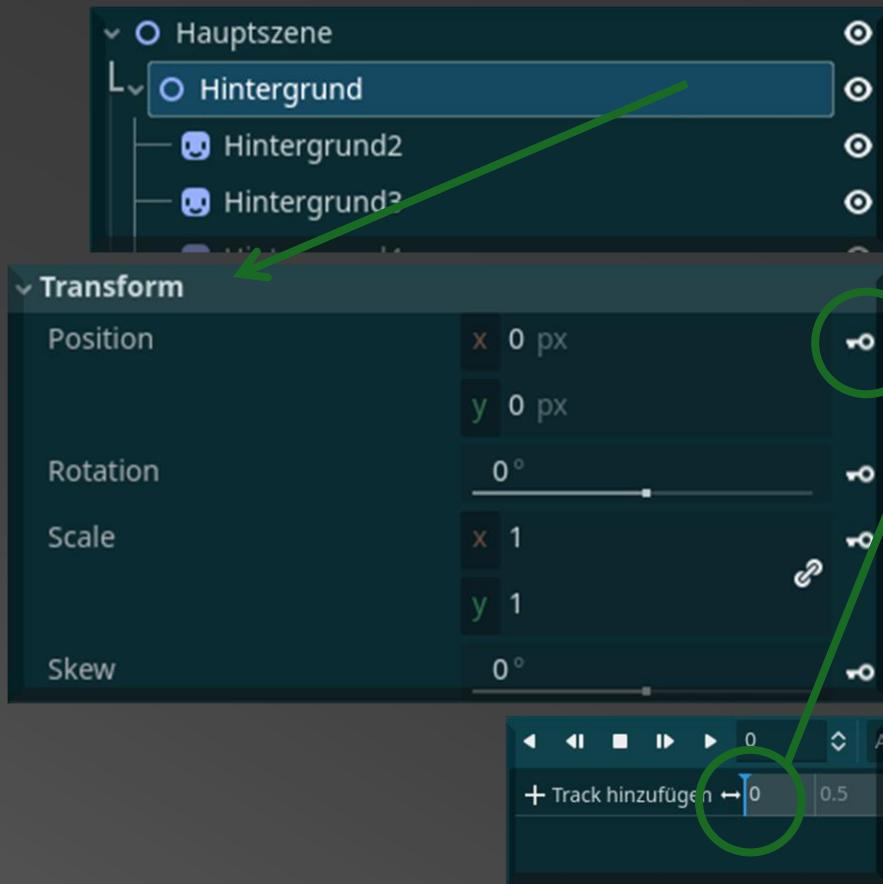
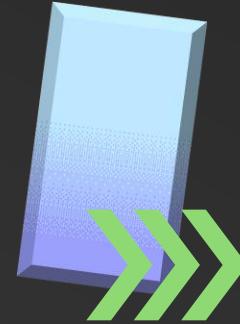
2. *Wiederholen* bei der Animation aktivieren

3. Die Dauer der Animation auf *8 Sekunden* setzen

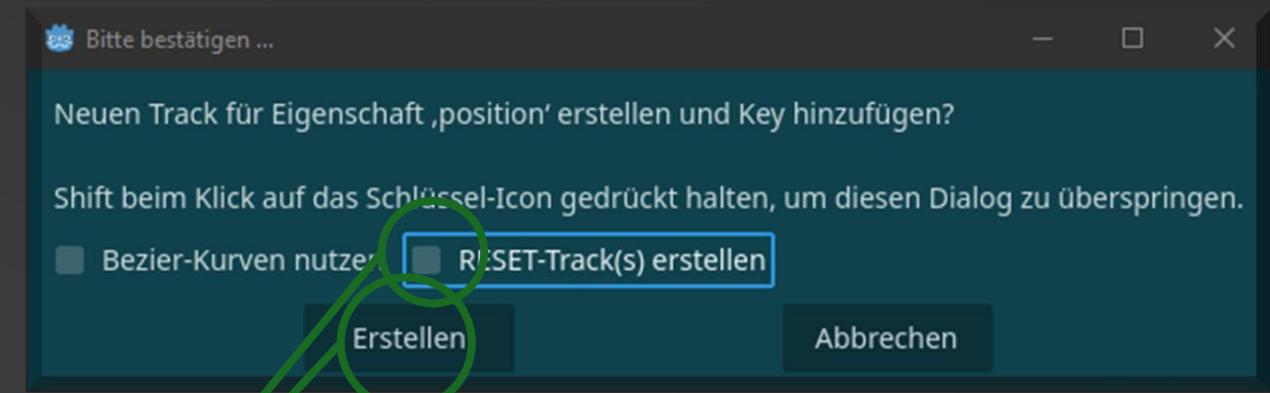
Animation»»

- ❖ Animationen bestehen aus *Keyframes*
- ❖ Ein Keyframe speichert einen Wert einer Variable zu einer bestimmten Zeit (z.B. Position)

Hintergrund- Animation



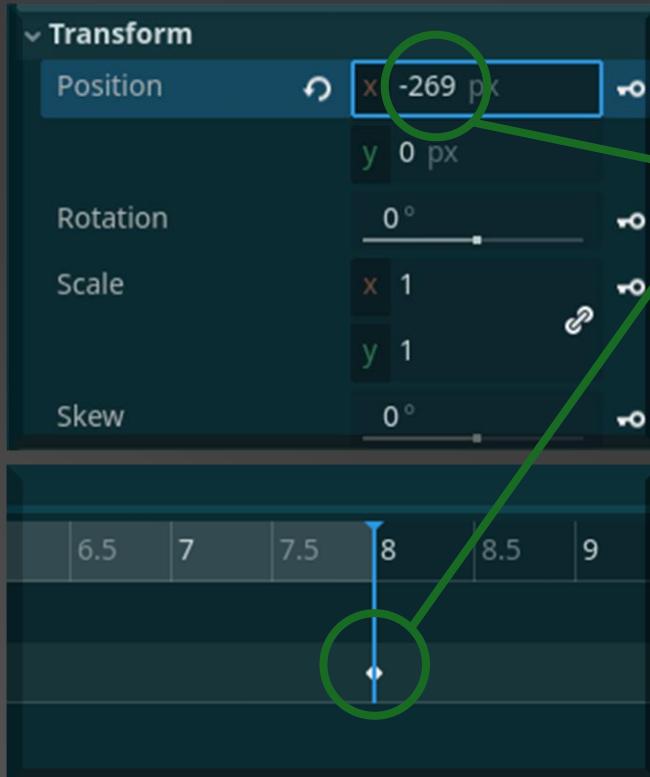
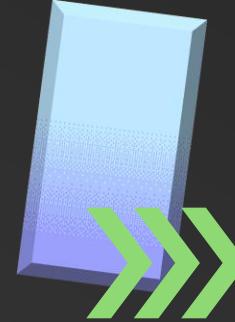
1. Bei Sekunde 0 in der Animation die X-Position des Hintergrund-Nodes auf 0 setzen



2. Ein Keyframe ohne Reset-Track erstellen

→ Reset-Tracks sorgen bei mir manchmal für komische Bugs (wahrscheinlich ein Anwenderfehler von mir)

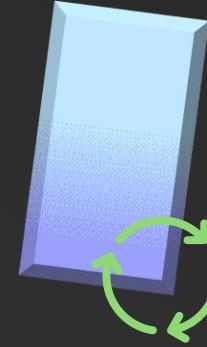
Hintergrund- Animation



1. Bei Sekunde 8 in der Animation die X-Position des Hintergrund-Nodes auf -269 setzen

2. Das Spiel testen!

Hintergrund – Das haben wir geschafft

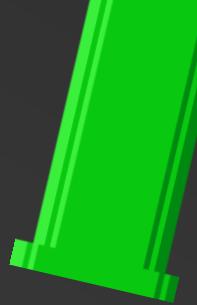


- ❖ Das Spiel hat jetzt einen Hintergrund
- ❖ Der Hintergrund bewegt sich

Spieleentwicklungs-Tipp:

Es ist nicht alles so wie man denkt. Die intuitive Lösung ist nicht unbedingt die beste Lösung.
Bsp.: Der Hintergrund bewegt sich statt des Vogels.

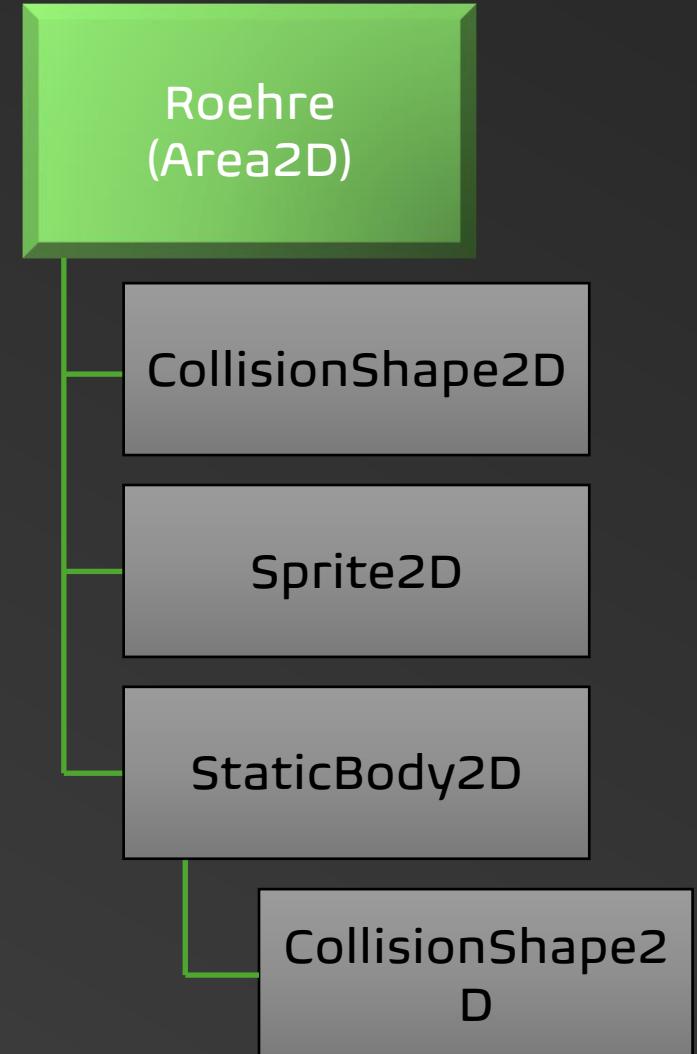
Röhren



- ❖ Neue 2D-Szene mit dem Namen *Roehren* erstellen
- ❖ Area2D Unternode hinzufügen
- ❖ Unternode in Roehre umbenennen

4. Unternodes hinzufügen

5. Speichern!

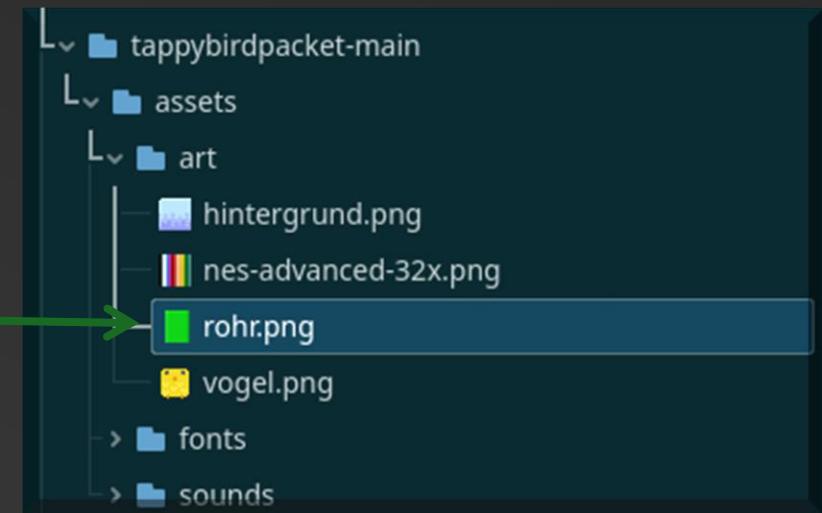
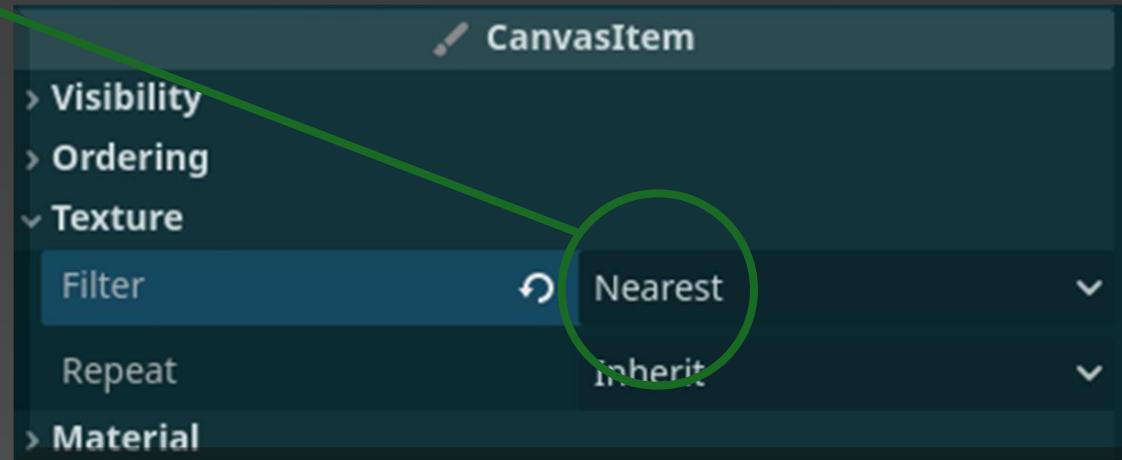


Röhren - Aussehen



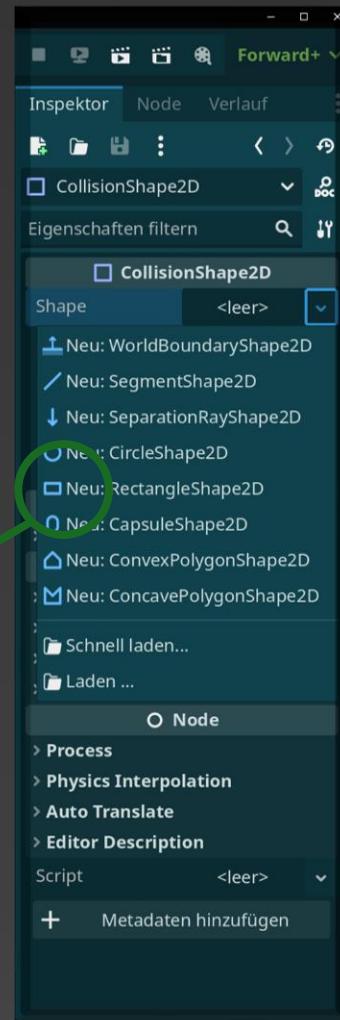
- ❖ Setze die Textur des Sprite2D auf *rohr.png*

2. Setze bei dem Haupt-Node *Filter* unter Texture auf Nearest

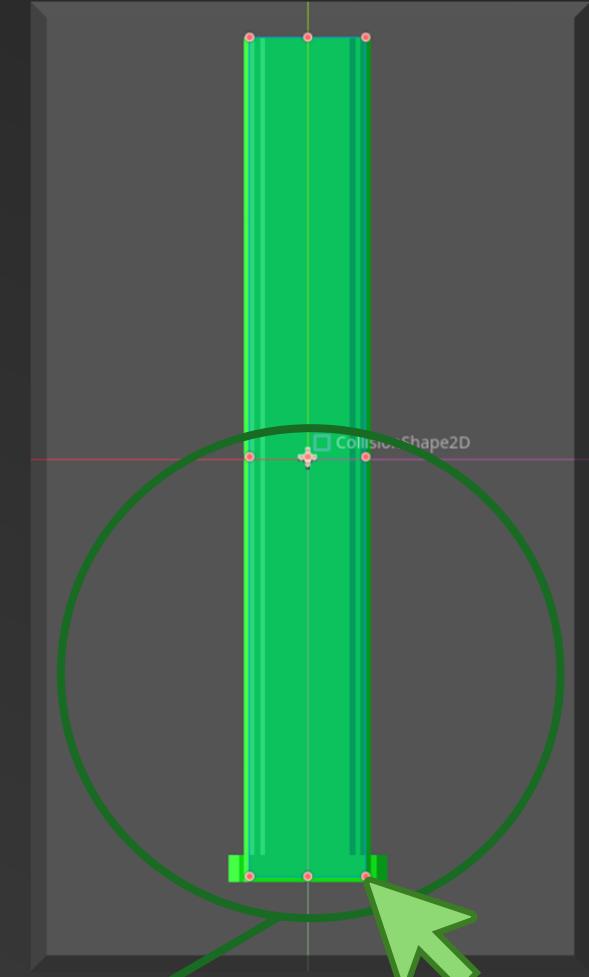


Röhren - Kollision

1. In der *ersten* CollisionShape2D eine neue RectangleShape2D erstellen



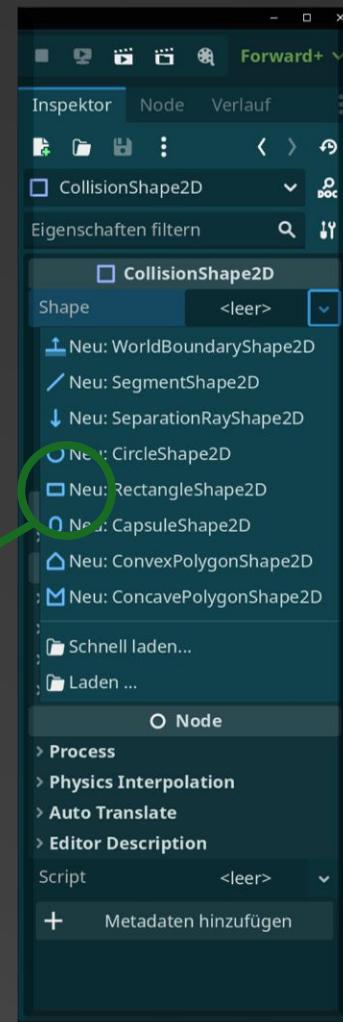
2. Größe auf Röhre einstellen



Einen Pixel kleiner als die Röhre!

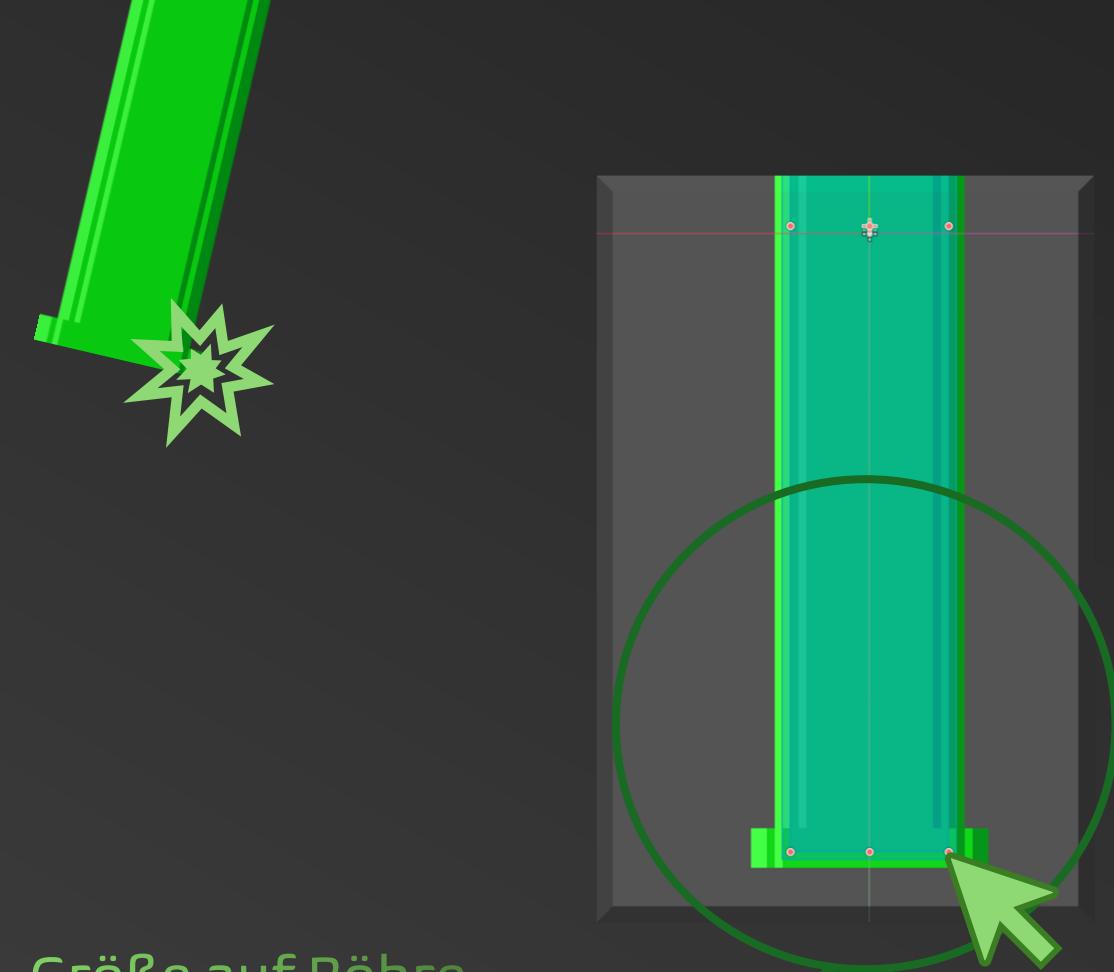
Röhren – Kollision #2

1. In der *zweiten* CollisionShape2D eine neue RectangleShape2D erstellen

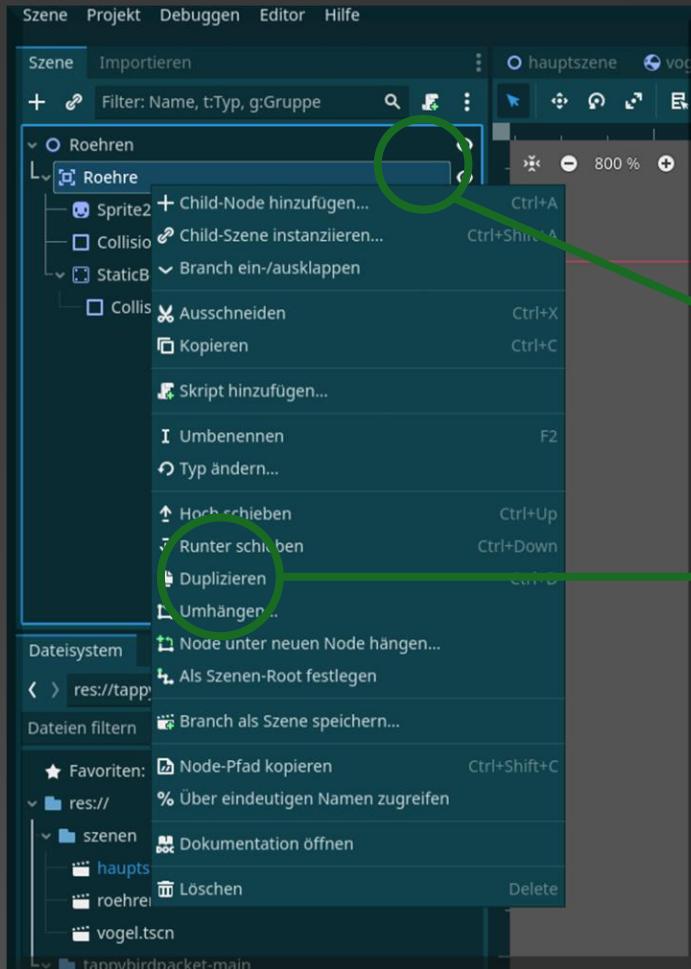


2. Größe auf Röhre einstellen

Zwei Pixel kleiner als die Röhre!

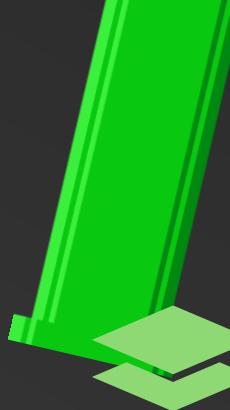


Röhren – Duplizieren

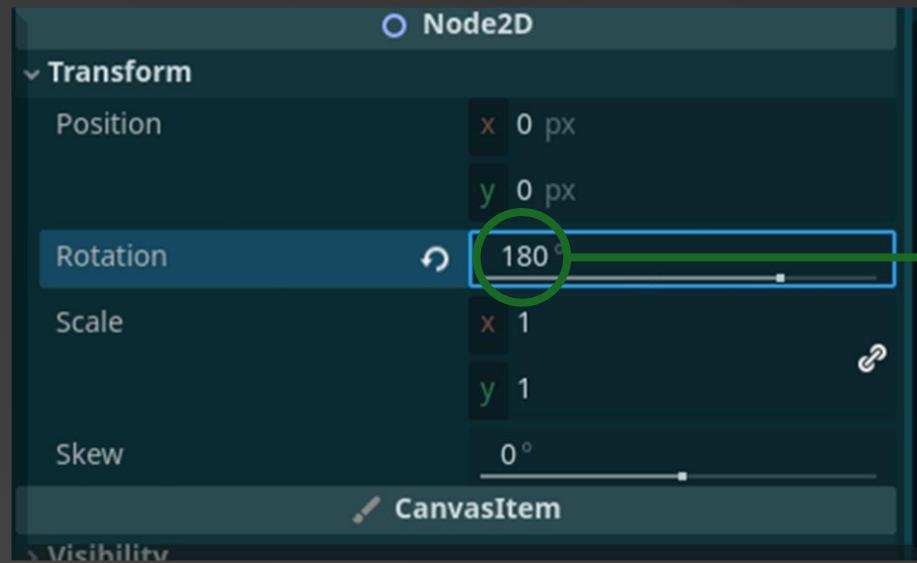


1. Die Area2D
rechtsklicken

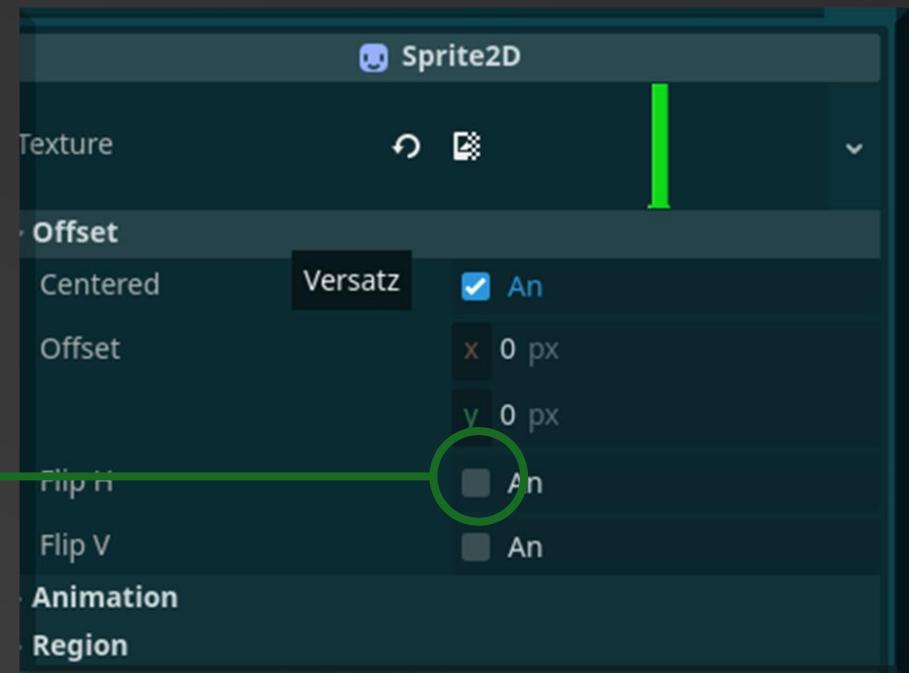
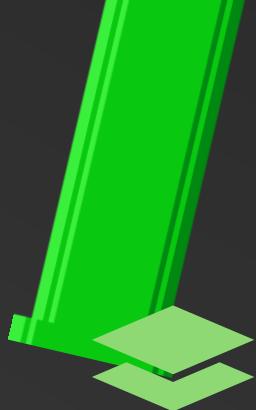
2. Die Area2D
duplizieren



Röhren – Aussehen

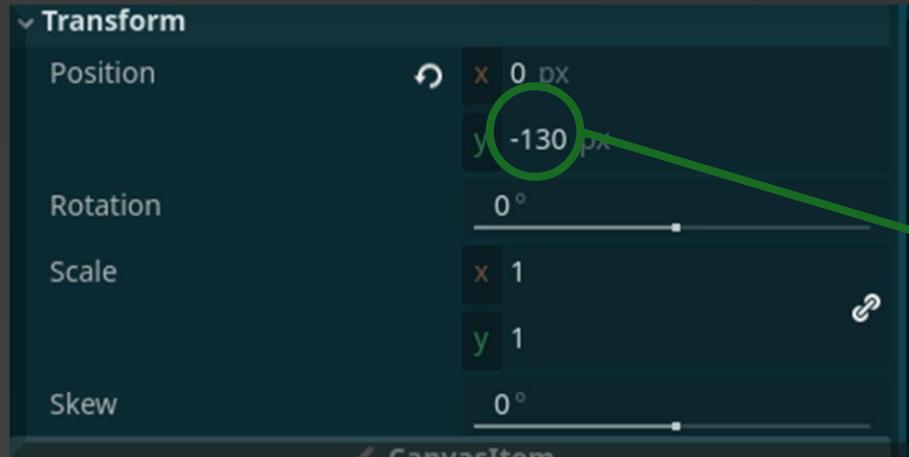


1. Unter Transform die Rotation von Roehre2 auf 180° setzen



2. Unter Offset Flip H von dem Sprite2D der Roehre2 auf An setzen

Röhren – Position



1. Setze die Y-Position von Roehre auf -130

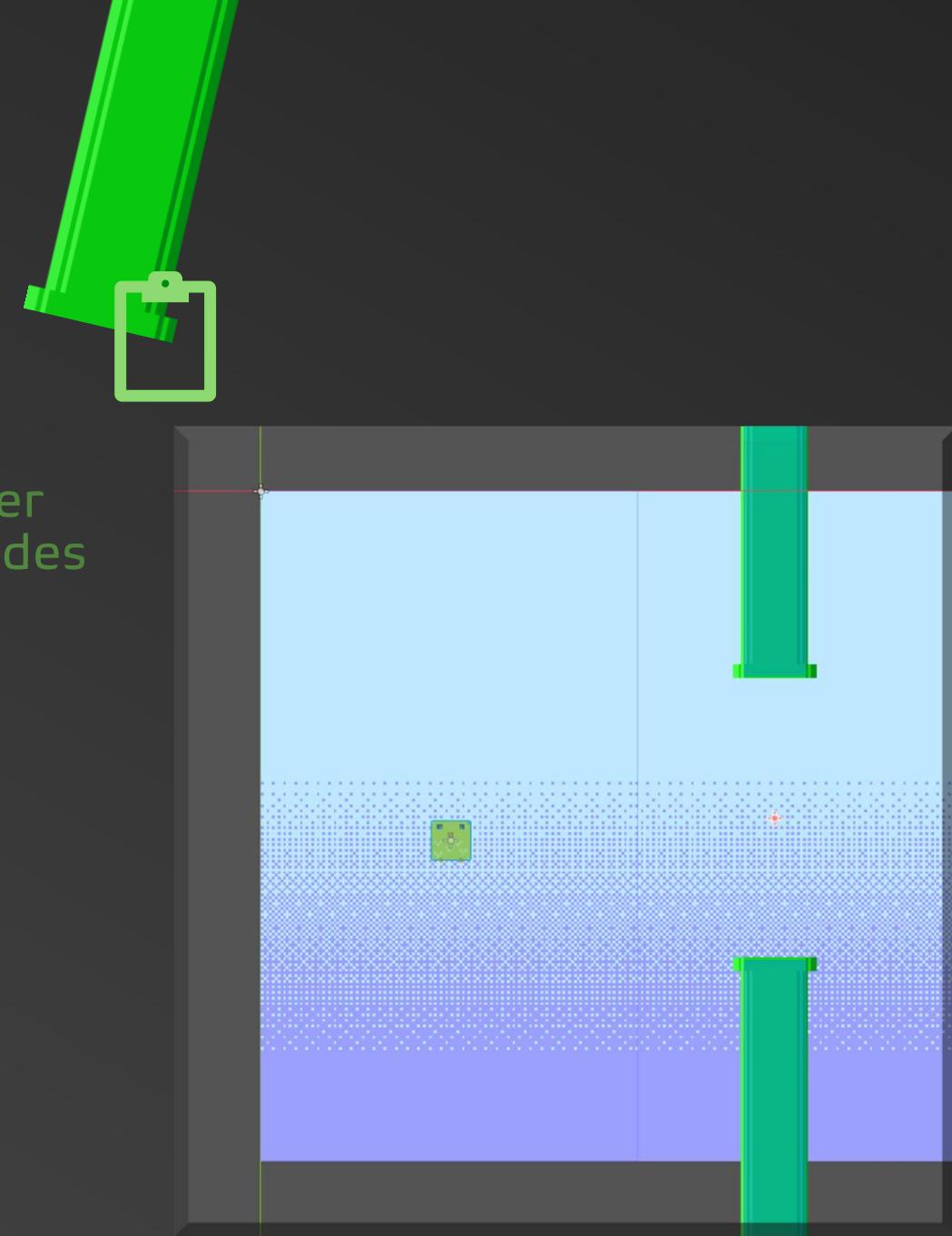
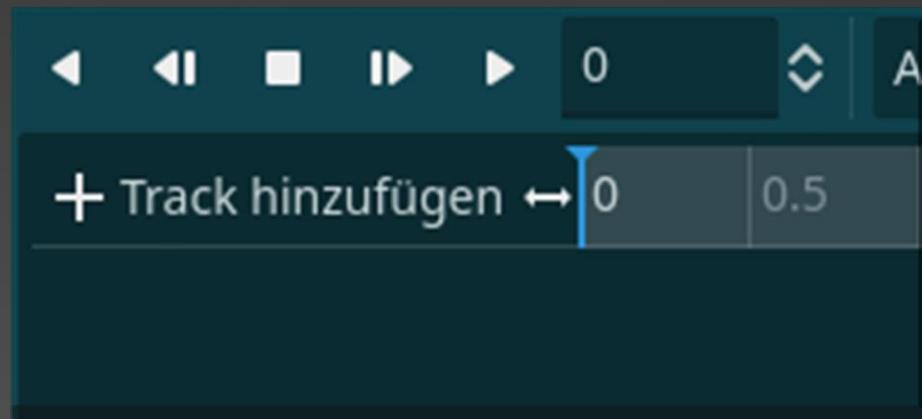


2. Setze die Y-Position von Roehre2 auf 130

3. Speichern!

Röhren – Einfügen

Füge die Röhren unter dem Hintergrund-Hauptnode so ein, dass sie bei Sekunde 0 in der Hintergrund-Animation grade so außerhalb des Bildschirms sind



Röhren – Das haben wir geschafft

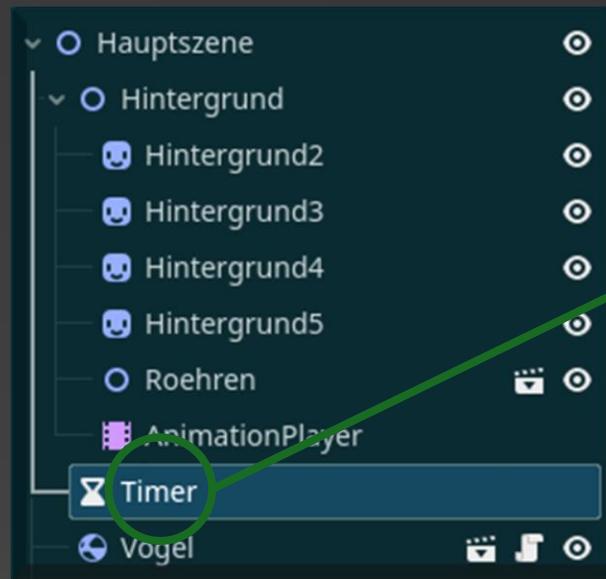


- ❖ Das Spiel hat jetzt Röhren
- ❖ Die Röhren bewegen sich mit dem Hintergrund

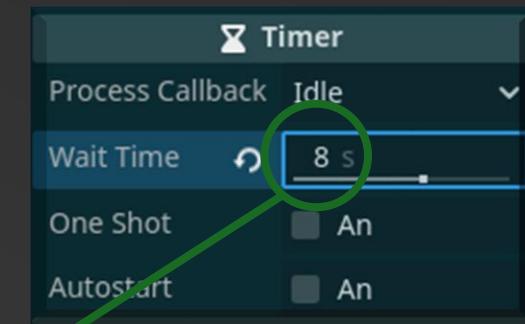
Spieleentwicklungs-Tipp:

Sei nett. Wenn die Spieler verlieren, soll es auch sichtbar sein.
Bsp.: Die Hit box der Röhre ist 1px kleiner als die Röhre.

Zufällig platzierte Röhren



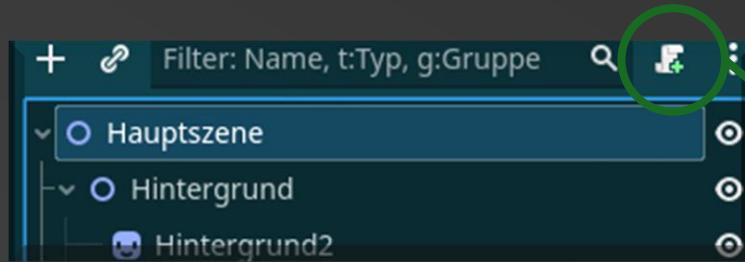
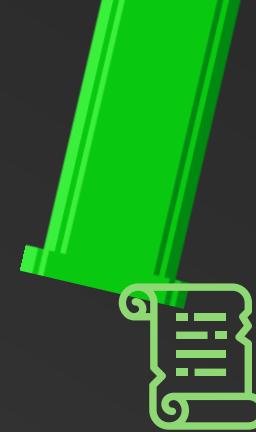
1. Erstelle einen Timer-Node in der Hauptszene



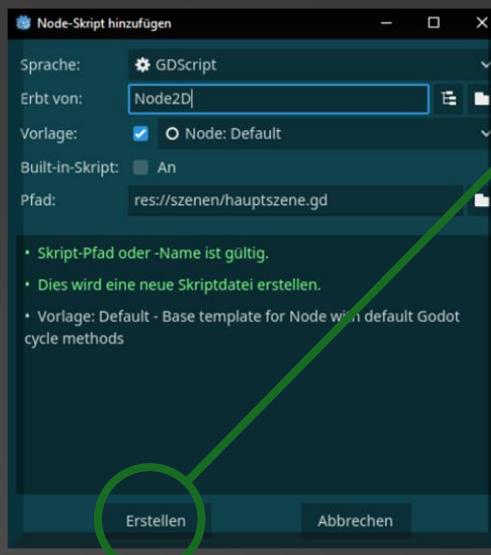
2. Setze Wait Time auf 8s

↖ 8 Sekunden weil die Animation sich danach wiederholt

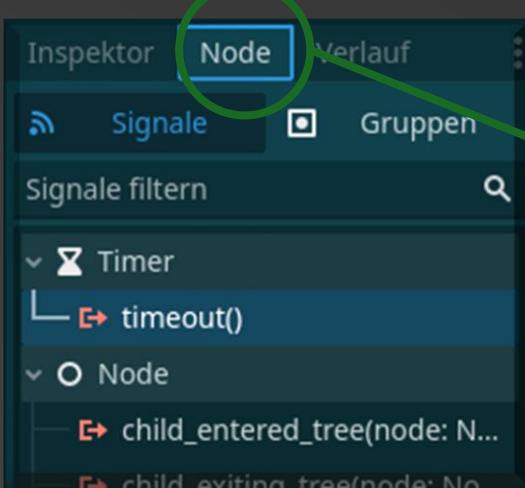
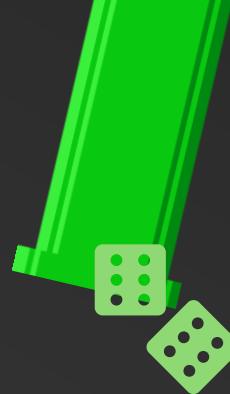
Zufällig platzierte Röhren



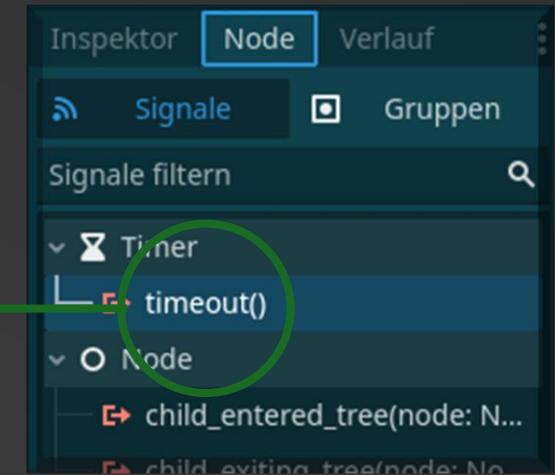
Erstelle ein neues Skript in der
Hauptszene



Zufällig platzierte Röhren

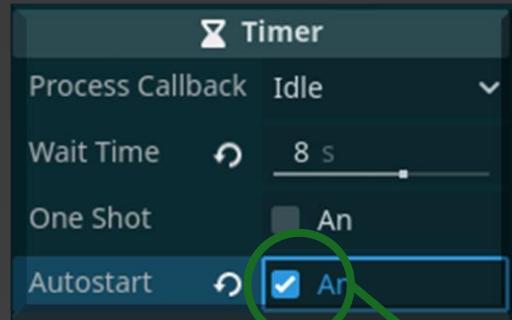


Öffne den *Node-Tab* vom Timer (oben rechts)



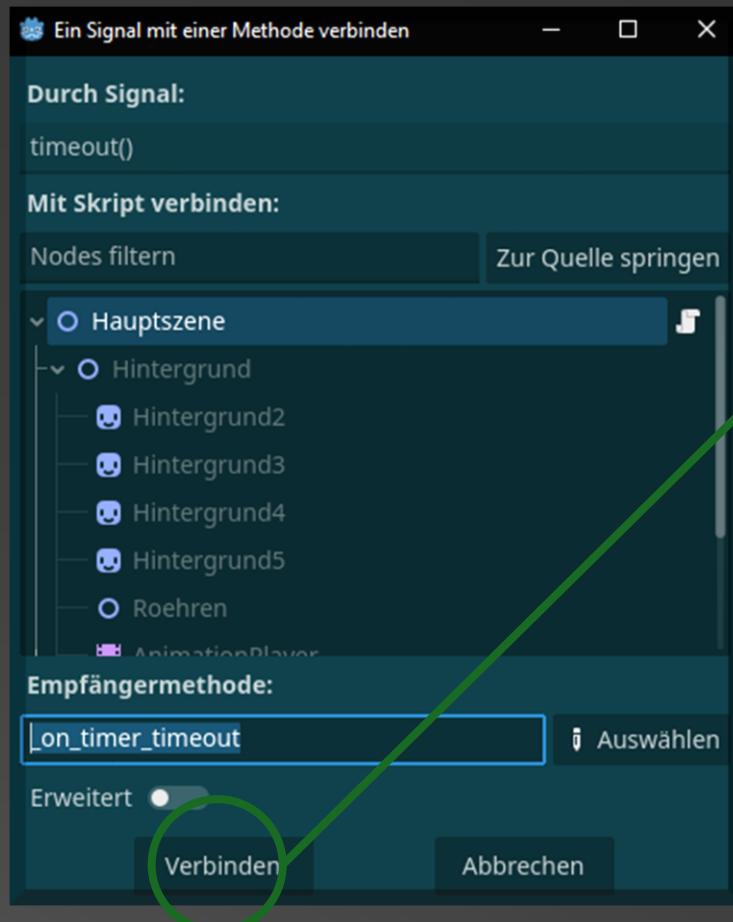
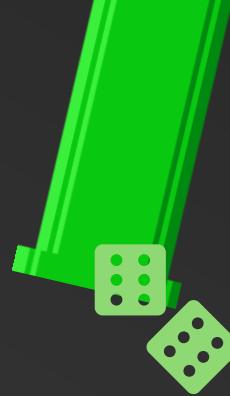
Doppelklicke auf *timeout()*

Zufällig platzierte Röhren



Aktiviere *Autostart* bei dem Timer

Zufällig platzierte Röhren



Verbinde dieses Signal mit dem gerade erstellten Skript

Zufällig platzierte Röhren - Code



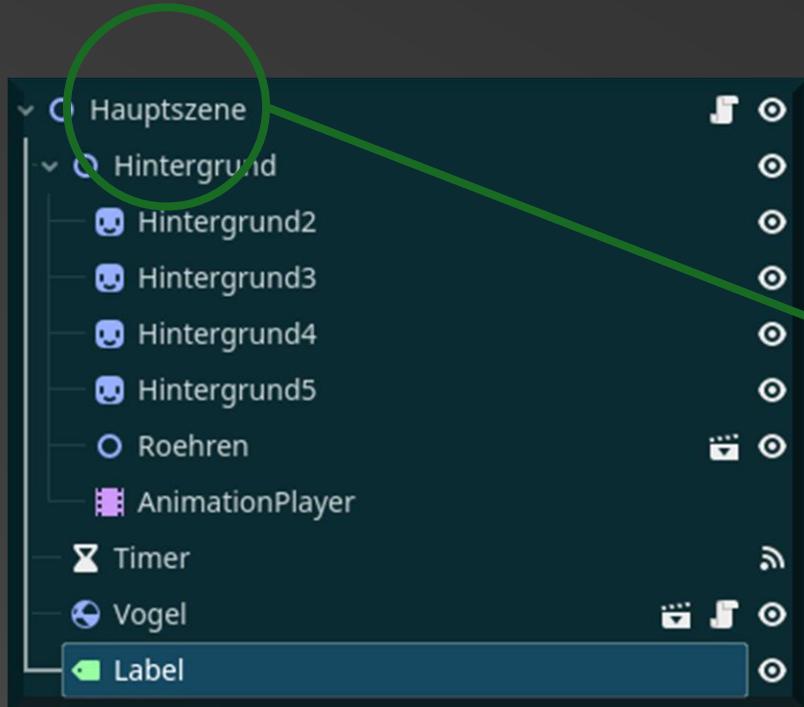
```
1  extends Node2D
2
3  func _on_timer_timeout() -> void: #Wird aufgerufen, wenn die Zeit abgelaufen ist
4      $Timer.start() #Timer neustarten
5      randomize() #Neue Zufallswerte generieren
6      $Hintergrund/Roehren.position.y = randf_range(60.0,175.0) #Position der Röhren auf einen Zuffalwert setzen
7
```

Zufällig platzierte Röhren - Geschafft



- ❖ Die Röhren werden jetzt zufällig platziert

Verloren - Text



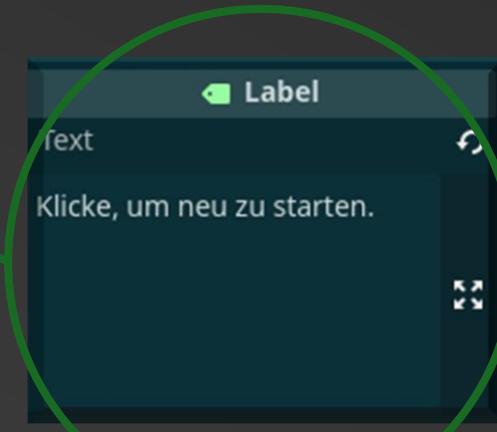
1. Erstelle ein neues *Label* in der Hauptszene

2. Nenne das Label *NeustartText*

Verloren - Text



1. Setze den Text des Labels auf
Klicke, um neu zu starten.



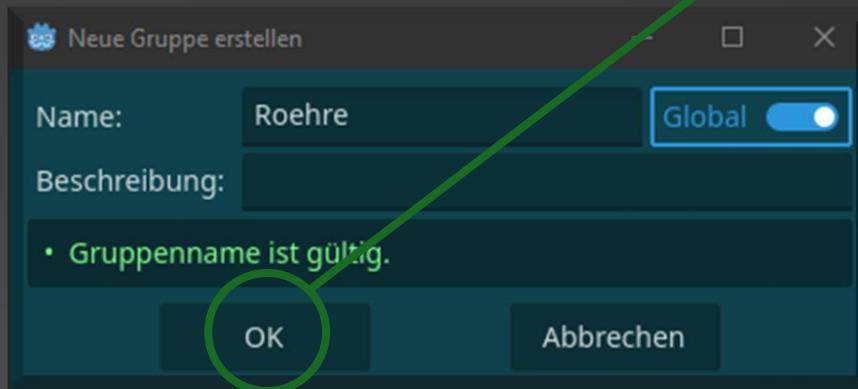
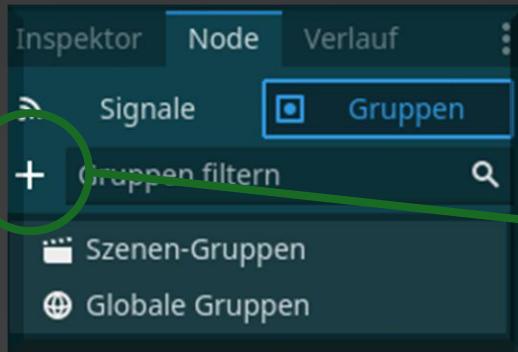
2. Verstecke das Label



Verloren - Theorie

- ❖ Der Vogel berührt eine Röhre und sendet ein Signal
- ❖ Der Vogel schaltet das Springen aus
- ❖ Die Hauptszene empfängt das Signal
- ❖ Der Neustart-Text wird angezeigt
- ❖ Beim Klicken startet das Spiel neu

Verloren - Kollision

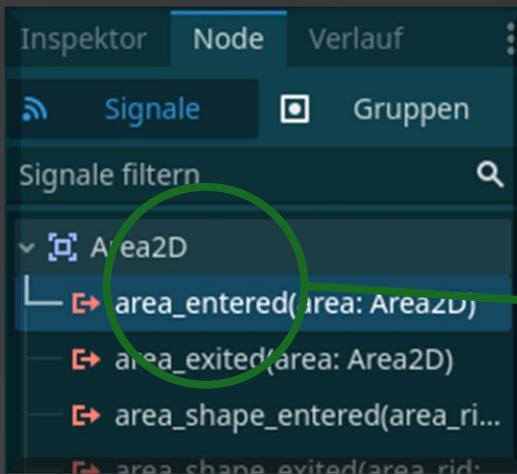


1. Füge im Node-Tab von einem der Röhren(Area2D) in der Roehren-Szene eine *globale* Gruppe mit dem Namen *Roehre* hinzu

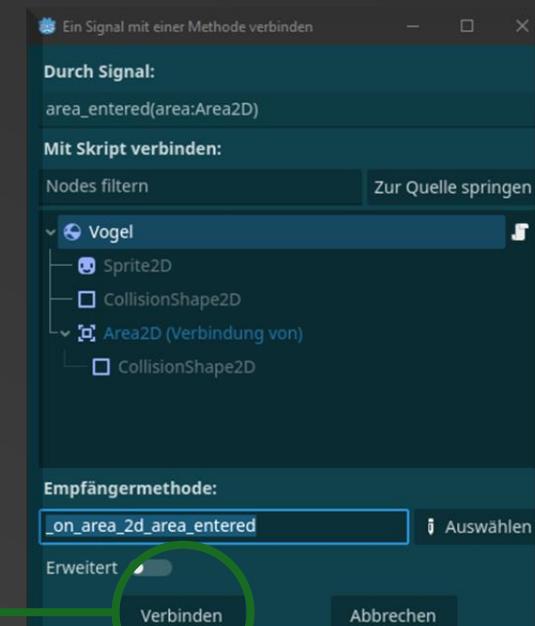
2. Aktiviere die *Roehre* Gruppe bei beiden Röhren(Area2d)



Verloren – Vogel Kollision



1. In der Area2D des Vogels das Signal *area_entered* doppelklicken

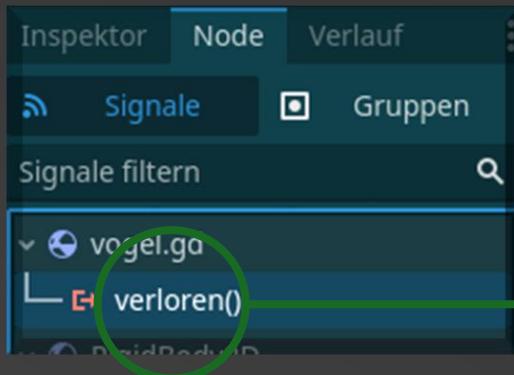


2. Verbinde das Signal mit dem Skript des Vogels

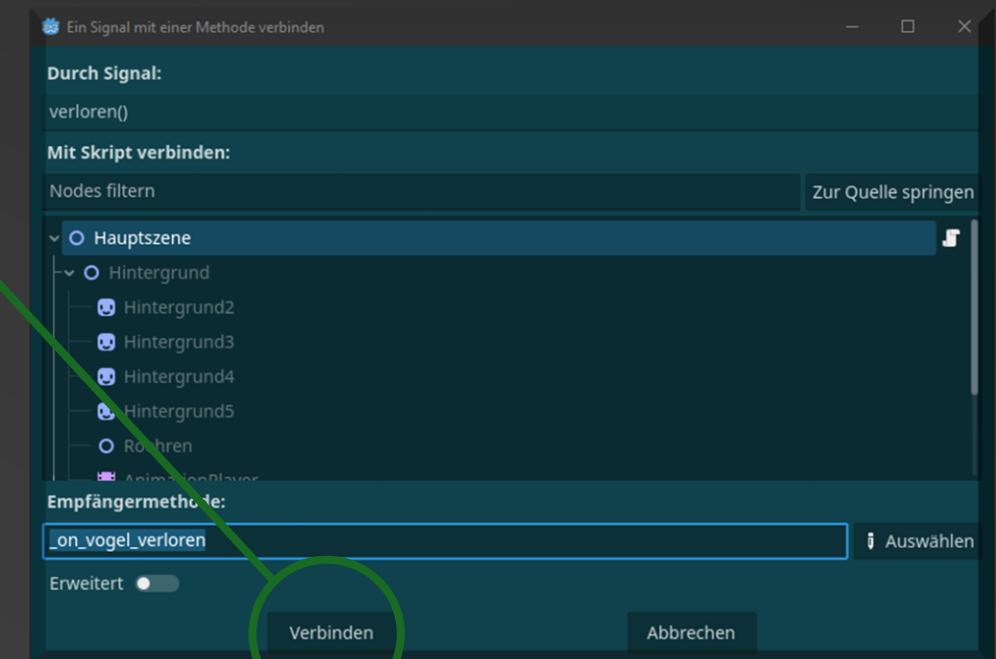
Verloren – Vogel Skript erweitern

```
1  extends RigidBody2D
2  signal verloren
3
4  @export var JUMP_HEIGHT: float = 50.0 #Sprunghöhe
5  const UP_VECTOR: Vector2 = Vector2.UP #Richtung, die nach Oben zeigt
6  var can_jump: bool = true #Gibt an, ob der Vogel springen kann
7
8  ↳ func _input(event: InputEvent) -> void: #Wird bei einer Eingabe aufgerufen
9    ↳ if event.is_action_pressed("click"): #Wenn geklickt wird
10      ↳ if can_jump: #Schaut, ob der Vogel springen kann
11        ↳ apply_impulse(UP_VECTOR * JUMP_HEIGHT) #Vogel nach oben werfen
12
13
14  ↳ func _on_area_2d_area_entered(area: Area2D) -> void: #Wird bei der Kollision mit einer Area2D aufgerufen
15    ↳ if area.is_in_group("Roehre"): #Schaut, ob die Area2D in der Roehre-Gruppe ist
16      ↳ verloren.emit() #Sendet das verloren signal
17      ↳ can_jump = false #Sorgt dafür, dass der Vogel nicht mehr springen kann
18
```

Verloren – Vogel Signal verbinden



Das *Verloren-Signal* des Vogels in der
Hauptszene mit dem Skript in der
Hauptszene verbinden



Verloren – Hauptzonen-Skript erweitern



```
1  extends Node2D
2  var has_lost: bool = false #Speichert, ob der Spieler verloren hat
3
4  ↳ func _on_timer_timeout() -> void: #Wird aufgerufen, wenn die Zeit abgelaufen ist
5      $Timer.start() #Timer neustarten
6      randomize() #Neue Zufallswerte generieren
7      $Hintergrund/Roehren.position.y = randf_range(60.0,175.0) #Position der Röhren auf einen Zufalls Wert setzen
8
9
10 ↳ func _on_vogel_verloren() -> void: #Wird aufgerufen, wenn der Spieler verloren hat
11     $Hintergrund/AnimationPlayer.speed_scale = 0.0 #Stoppt die Hintergrund-Bewegung
12     $NeustartText.show() #Zeigt den Neustart-text an
13     has_lost = true #Aktiviert den Verloren-Zustand
14
15 ↳ func _input(event: InputEvent) -> void: #Wird aufgerufen, wenn eine Eingabe getätigt wird
16     ↳ if event.is_action_pressed("click"): #Wird aufgerufen, wenn geklickt wird
17         ↳ if has_lost: #Schaut ob der Spieler verloren hat
18             ↳ get_tree().reload_current_scene() #Lädt die Szene neu
19 |
```

Verloren – Das haben wir geschafft

- ❖ Wenn man eine Röhre berührt, verliert man
- ❖ Man kann das Spiel neu starten

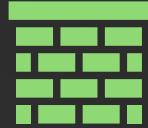
Godot-Wissen:

Signale können auch selbst im Code erstellt werden.

Selbst erstellte Signale kann man genau so wie vorgegebene Signale benutzen.
So konnten wir eine vorgegebene Godot-Mechanik mit selbst erstellten Inhalten benutzen.

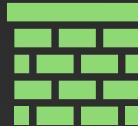
Das habe ich am Anfang gemeint.

Begrenzungen – Das Problem

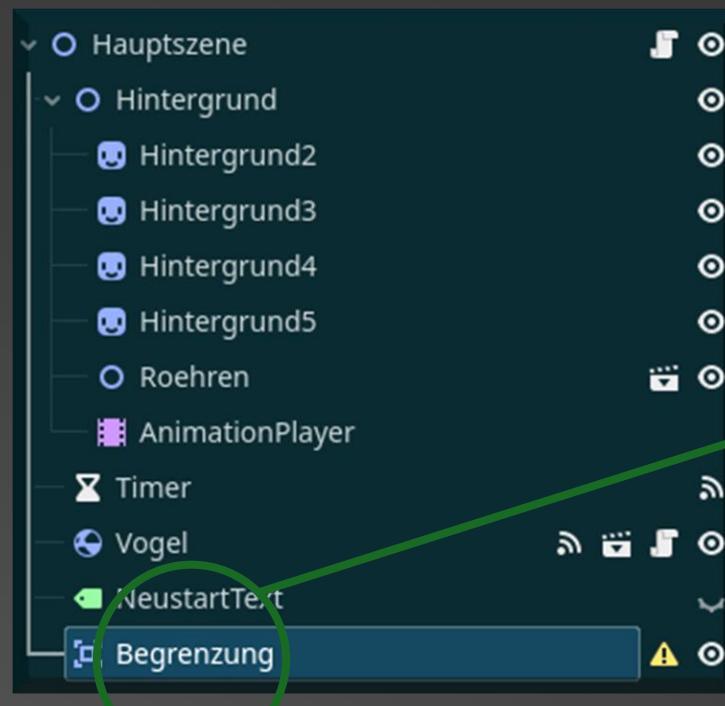


- ❖ Der Vogel kann immer noch außerhalb des Bildschirms springen
→ wir brauchen Bildschirm-Begrenzungen

Begrenzungen – Umsetzung

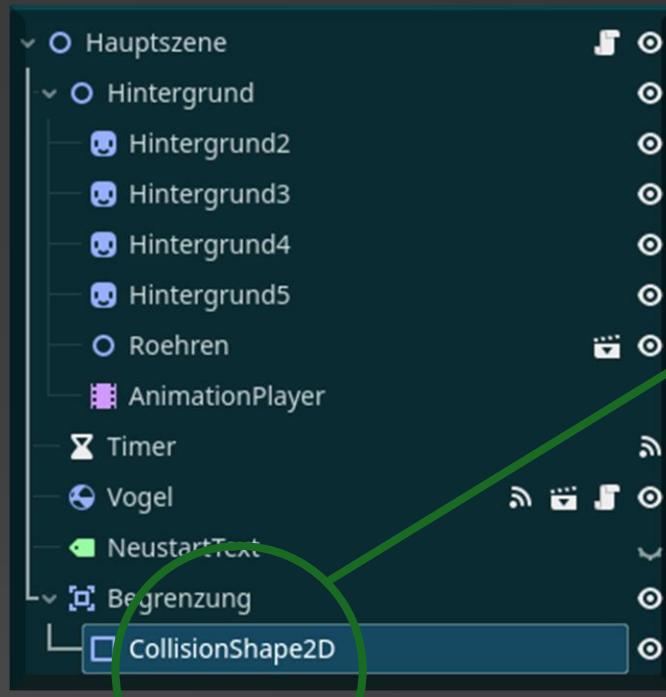


- ❖ Um eine Begrenzung zu bauen, brauchen wir nur eine Area2D, die in der Roehren-Gruppe ist



1. Füge der Hauptszene eine *Area2D* hinzu
2. Nenne die *Area2D Begrenzung*

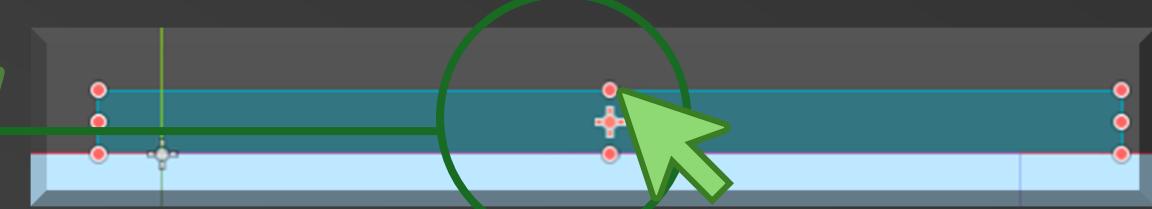
Begrenzungen – Umsetzung



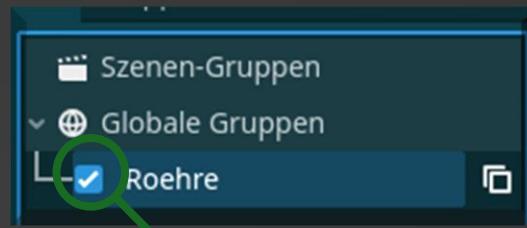
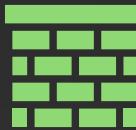
1. Füge dem *Begrenzung-Node* eine *CollisionShape2D* hinzu

2. Erstelle in der *CollisionShape2D* eine *RectangleShape2D*

3. Mache mit der *RectangleShape2D* einen *Streifen*



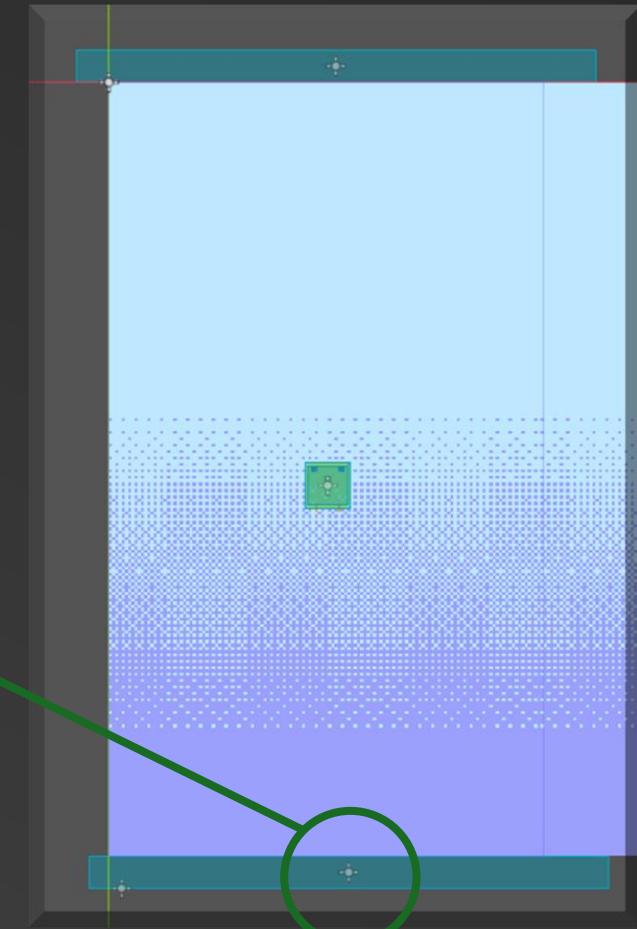
Begrenzungen – Umsetzung



1. Aktiviere bei der Area2D die **Roehre** Gruppe

2. Dupliziere die Area2D

3. Schiebe die zweite Area auf die gegenüberliegende Bildschirmseite



Begrenzungen – Das haben wir geschafft



- ❖ Wenn man den Bildschirmrand berührt, verliert man

Spieleentwicklungs-Tipp:

Es gibt keinen Grund Mechaniken doppelt zu programmieren.
Verwende alten Code wieder.

Bsp.: Wir haben die Roehren Gruppe auch für die Begrenzungen benutzt.

SFX – Irgendetwas fehlt... 🔊

- ❖ Das waren die 4 Teile eines Spiels:



IOIO
IOIO



- ❖ Diese haben wir umgesetzt:



IOIO
IOIO

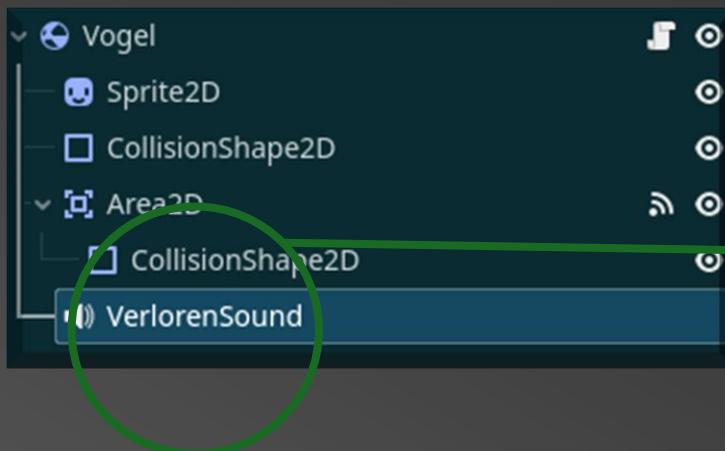


- ❖ Wir haben noch keinen Ton eingebaut.

SFX – Verlieren

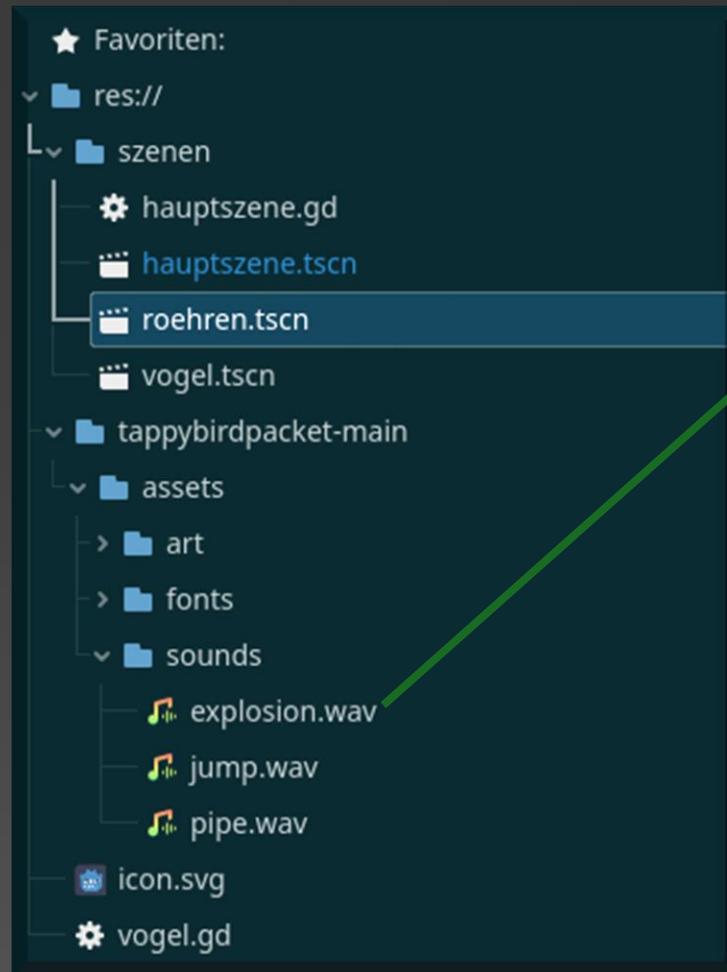


1. Erstelle in der Vogel-Szene einen neuen *AudioStreamPlayer*

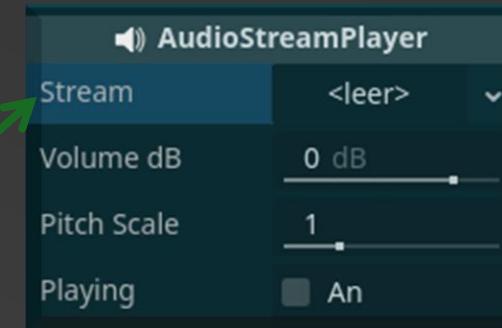


2. Nenne den *AudioStreamPlayer* *VerlorenSound*

SFX – Verlieren



1. Setze den *Stream* im
AudioStreamPlayer auf *explosion.wav*



2. Setze *Volume dB* im
AudioStreamPlayer auf -8 dB

SFX – Verlieren (Code im Vogel)

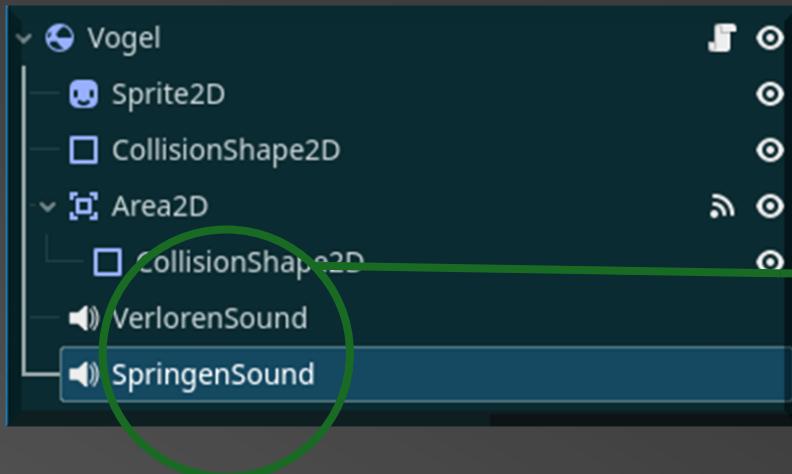


```
1  extends RigidBody2D
2  signal verloren
3
4  @export var JUMP_HEIGHT: float = 50.0 #Sprunghöhe
5  const UP_VECTOR: Vector2 = Vector2.UP #Richtung, die nach Oben zeigt
6  var can_jump: bool = true #Gibt an, ob der Vogel springen kann
7
8  ↳ func _input(event: InputEvent) -> void: #Wird bei einer Eingabe aufgerufen
9    if event.is_action_pressed("click"): #Wenn geklickt wird
10      if can_jump: #Schaut, ob der Vogel springen kann
11        apply_impulse(UP_VECTOR * JUMP_HEIGHT) #Vogel nach oben werfen
12
13
14  ↳ func _on_area_2d_area_entered(area: Area2D) -> void: #Wird bei der Kollision mit einer Area2D aufgerufen
15    if area.is_in_group("Roehre"): #Schaut, ob die Area2D in der Roehre-Gruppe ist
16      $VerlorenSound.play() #Spielt den VerlorenSound ab
17      verloren.emit() #Sendet das verloren signal
18      can_jump = false #Sorgt dafür, dass der Vogel nicht mehr springen kann
19
```

SFX – Springen

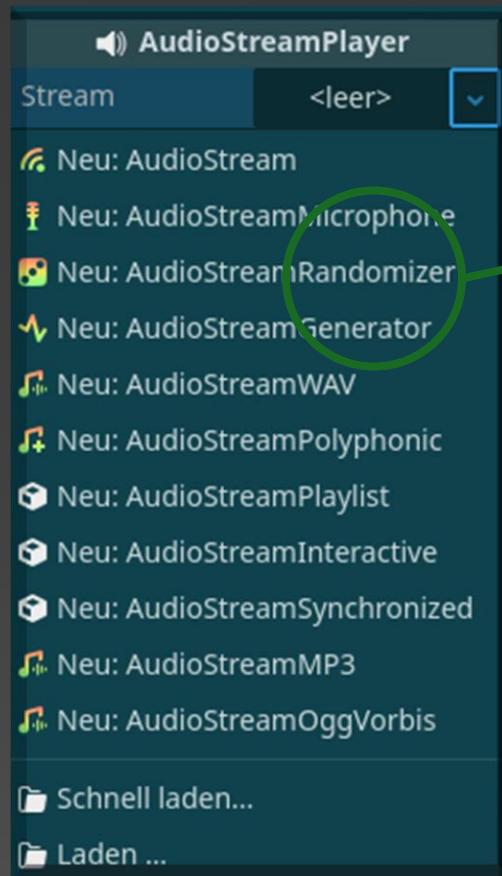


1. Erstelle in der Vogel-Szene einen zweiten *AudioStreamPlayer*



2. Nenne den *AudioStreamPlayer* diesmal *SpringenSound*

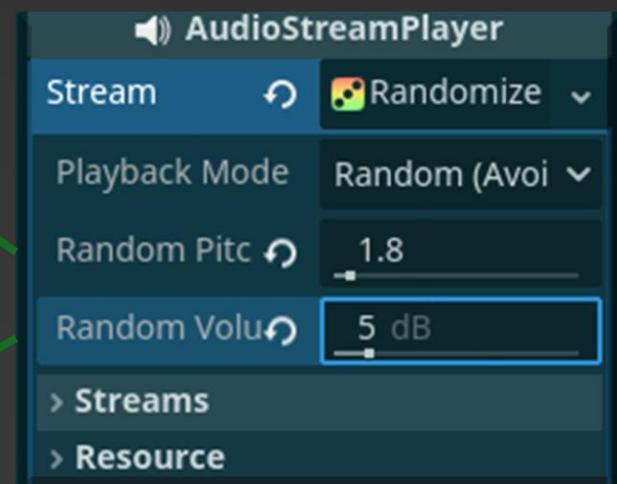
SFX – Springen



1. Erstelle in *SpringenSound* einen
neuen *AudioStreamRandomizer*

2. Setze *Random Pitch* auf 1.8

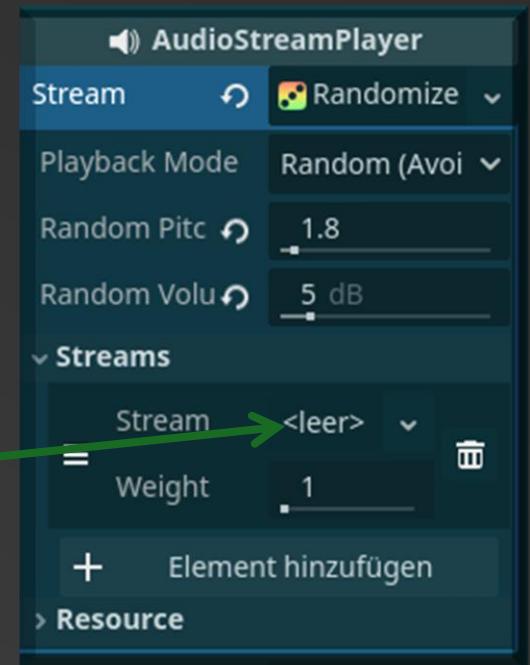
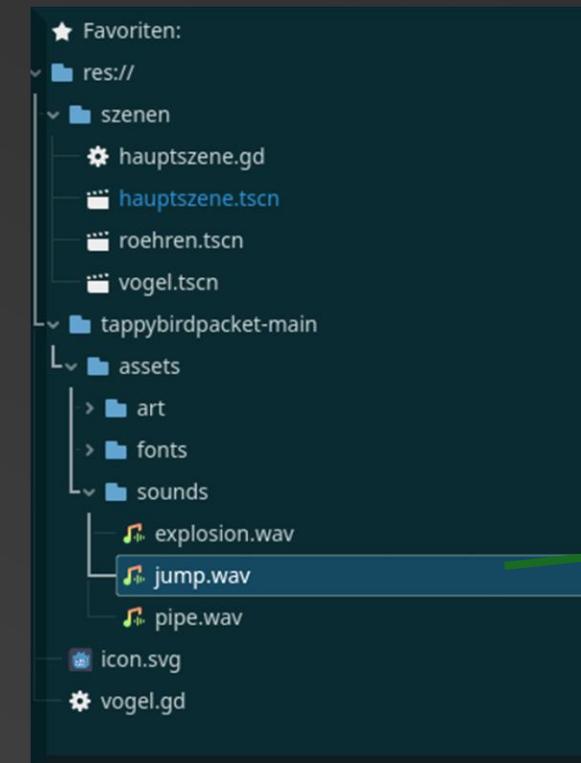
3. Setze *Random Volume* auf 5 dB



SFX – Springen



1. Füge Streams ein neues Element hinzu



2. Setze Stream auf jump.wav

SFX – Springen



1. Setze *Volume dB* auf -12 dB

SFX – Springen (Code im Vogel)

```
1  extends RigidBody2D
2  signal verloren
3
4  @export var JUMP_HEIGHT: float = 50.0 #Sprunghöhe
5  const UP_VECTOR: Vector2 = Vector2.UP #Richtung, die nach Oben zeigt
6  var can_jump: bool = true #Gibt an, ob der Vogel springen kann
7
8  ↳ func _input(event: InputEvent) -> void: #Wird bei einer Eingabe aufgerufen
9    ↳ if event.is_action_pressed("click"): #Wenn geklickt wird
10      ↳ if can_jump: #Schaut, ob der Vogel springen kann
11        ↳ $SpringenSound.play() #Spielt den Springen Sound ab
12        ↳ apply_impulse(UP_VECTOR * JUMP_HEIGHT) #Vogel nach oben werfen
13
14
15 ↳ func _on_area_2d_area_entered(area: Area2D) -> void: #Wird bei der Kollision mit einer Area2D aufgerufen
16    ↳ if area.is_in_group("Roehre"): #Schaut, ob die Area2D in der Roehre-Gruppe ist
17      ↳ $VerlorenSound.play() #Spielt den VerlorenSound ab
18      ↳ verloren.emit() #Sendet das verloren signal
19      ↳ can_jump = false #Sorgt dafür, dass der Vogel nicht mehr springen kann
20
```

SFX – Das haben wir geschafft

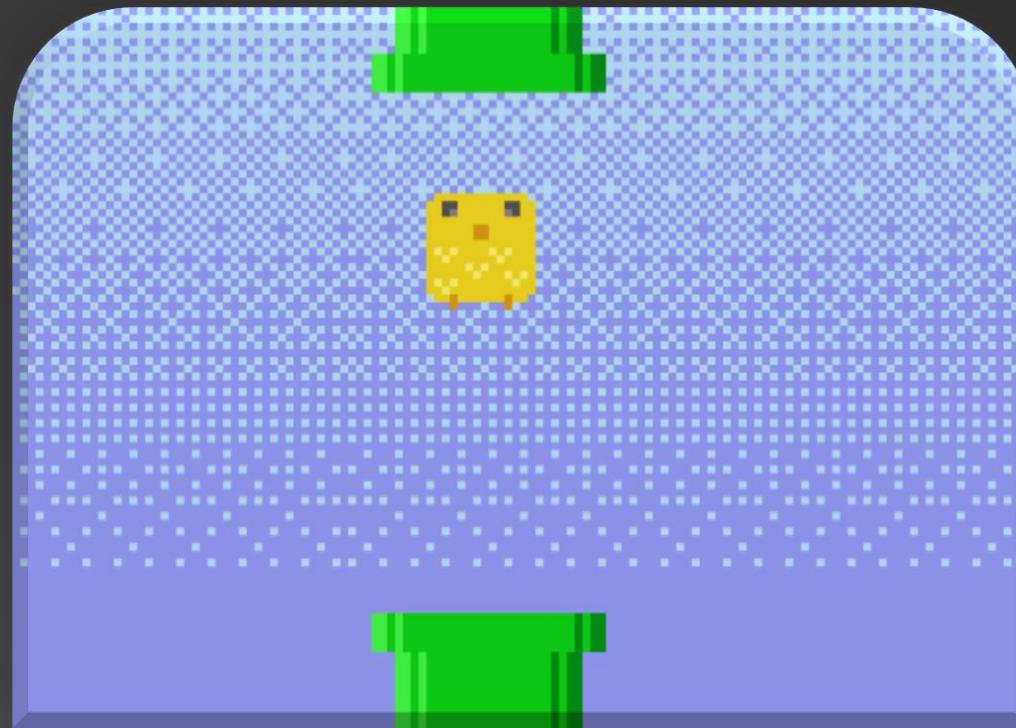
- ❖ Wenn man verliert, wird ein Ton abgespielt.
- ❖ Wenn man springt, wird auch ein Ton abgespielt.

Godot-Wissen:

AudioStreamPlayer können Geräusche abspielen.
Man kann auch eine zufällige Tonhöhe einstellen.

Das Spiel ist fertig!

❖ Glückwunsch! Du hast es geschafft das Spiel komplett fertig zu stellen.



Und jetzt?

Wenn du Spieleprogrammierung lernen möchtest,
findest du hier Lernmaterialien:

- ❖ https://kidscancode.org/godot_recipes/4.x/
- ❖ <https://www.gdquest.com/>
- ❖ <https://www.youtube.com/letsgamedev>



Viel Spaß beim *Entwickeln!*