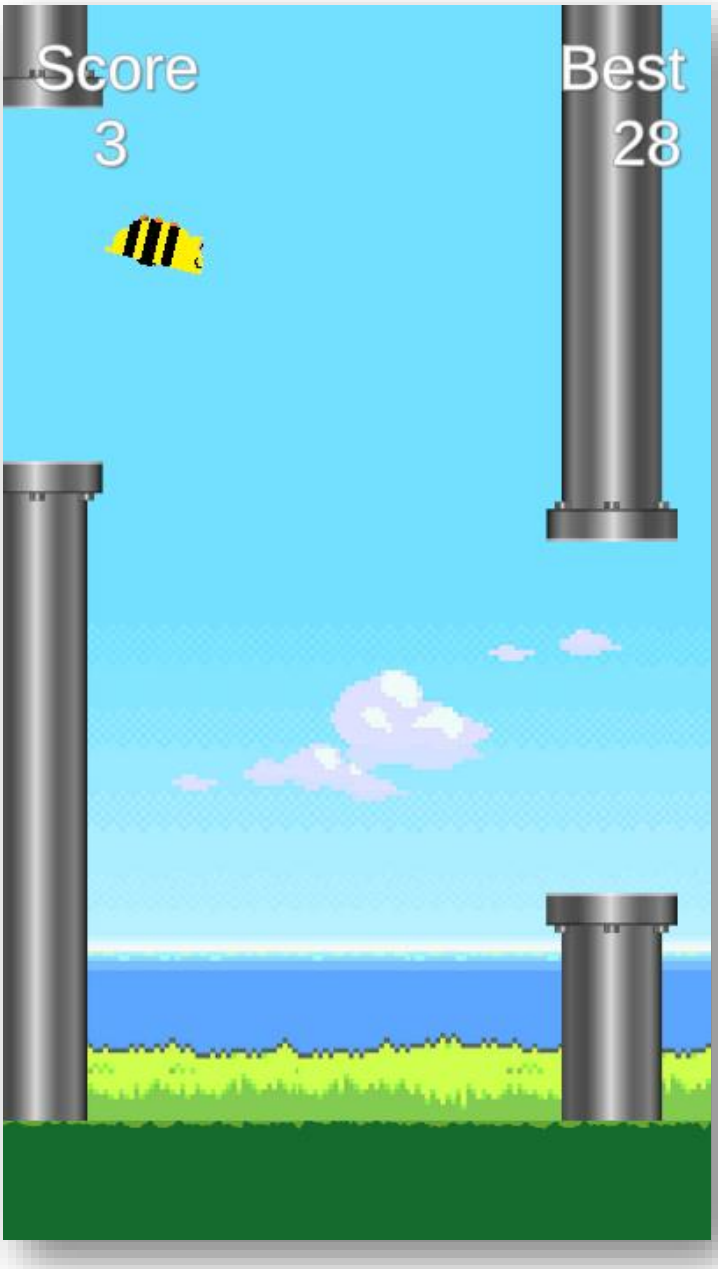
Unity – Zweite Einheit

In dieser Einheit:

Happy Bee



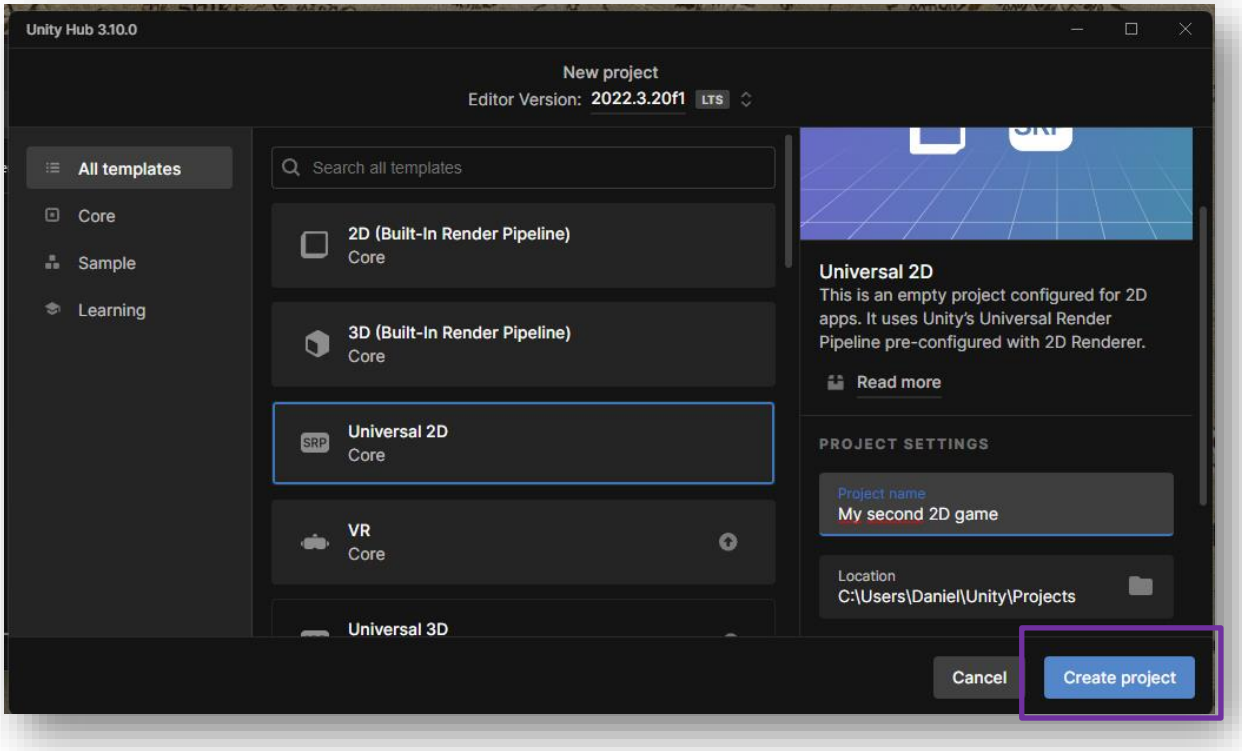
Inhalt

Willkommen zurück!

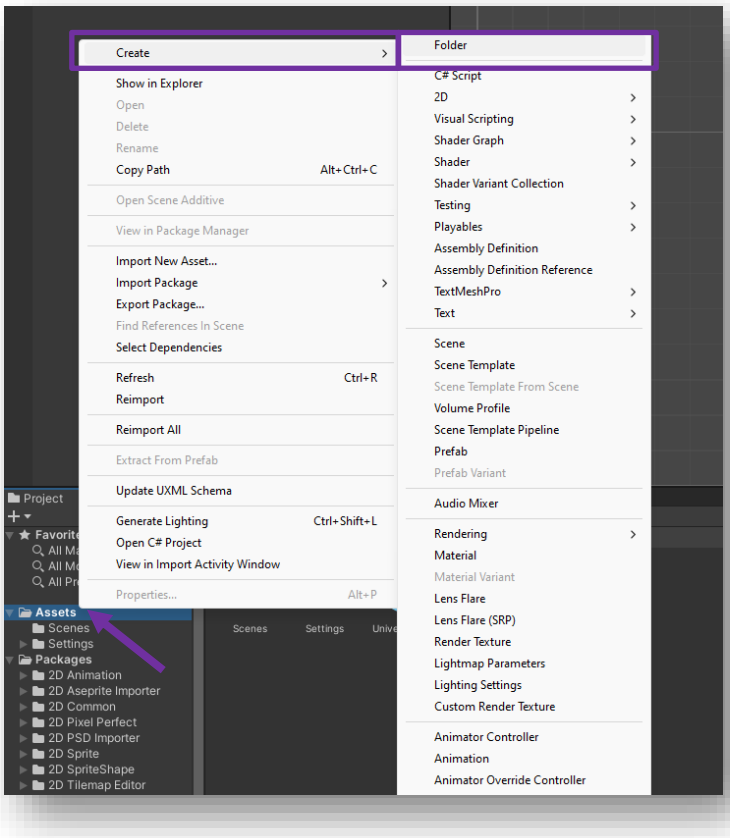
Hallo und herzlich willkommen zum **zweiten Teil der 2D Spieleentwicklung in Unity**.

Solltest du den ersten Teil (MO_I_2D-Unity - mein erstes 2D Spiel) noch nicht abgeschlossen haben, tu dies bitte um die Grundlagen der 2D Spieleentwicklung kennen zu lernen.

Wir erstellen nun ein neues Projekt und wählen diesmal die **Universal-Render Pipeline** (Universal 2D) aus.



Assets - Der Hintergrund (Background)

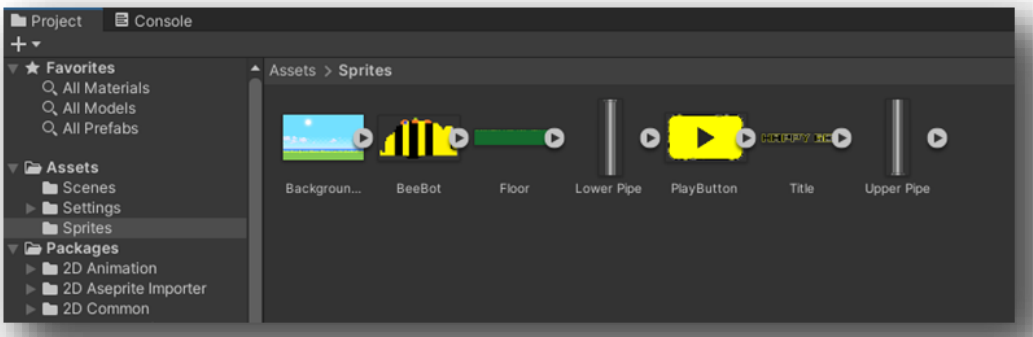


Assets sind in diesem Fall unsere Bilder. Es können aber auch andere Dateien darin zu finden sein, ähnlich wie Baublöcke um unser Projekt zu bauen.

Du erinnerst dich vielleicht noch an unser letztes Projekt. In diesem hatten wir in Assets unser Dreieck, da wir es nicht in der Hierarchy gebraucht haben, aber es später in einem Skript einsetzen wollten.

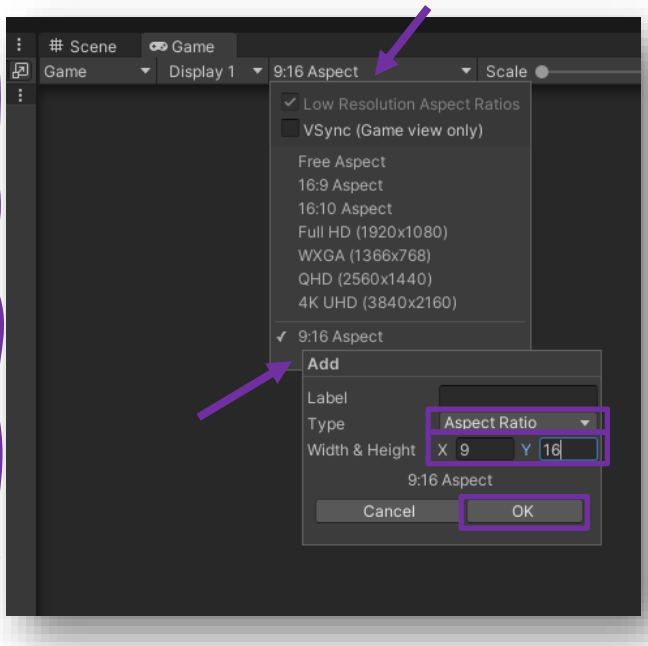
Wir beginnen damit, einen **neuen Ordner** unter „Assets“ (in Project) anzulegen. Dies können wir, indem wir auf **Assets rechtsklicken** und dann **Create** und **Folder** auswählen. Diesen nennen wir **Sprites**. Darin sind dann unsere ganzen Ressourcen wie Bilder zu finden.

Nun ziehen wir unsere Bilder (die findest du in der Materialbörse) in diesen Ordner, dieser sollte dann so aussehen:

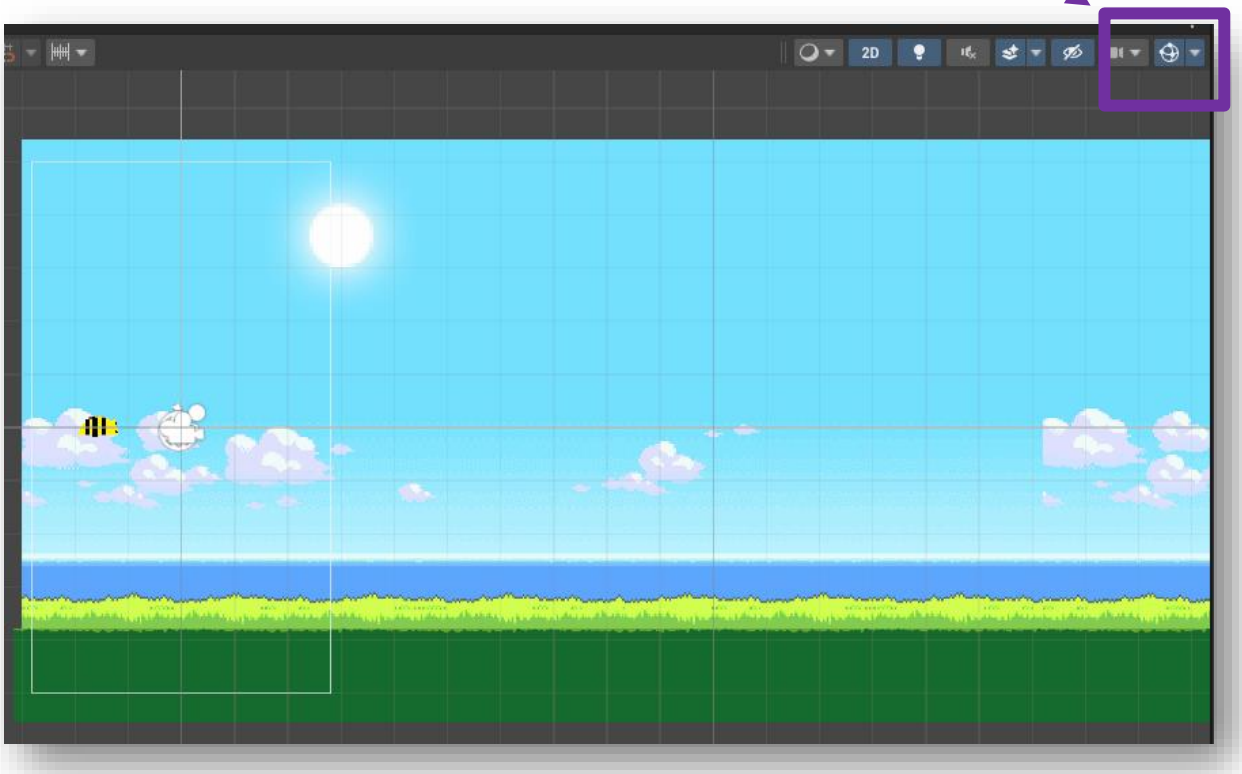


Als nächstes wechseln wir in der Scene auf Game (das ist der Reiter neben Scene) und lassen uns die Ansicht während des Spiels anzeigen. Hier ändern wir nun die Größe des Spiels.

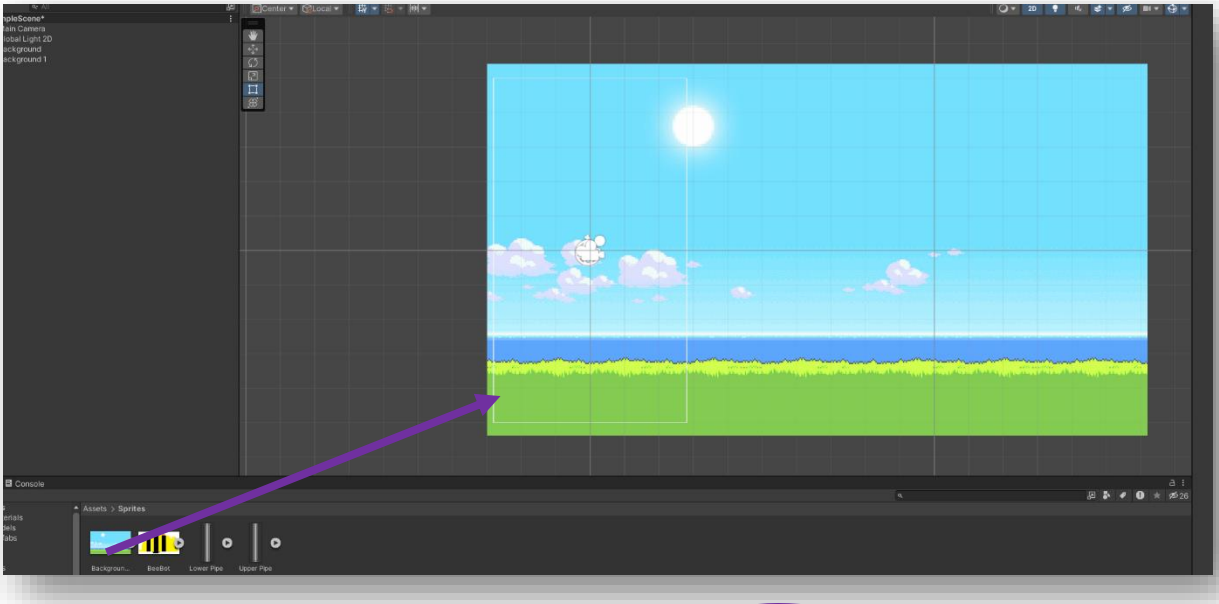
Dazu drücken wir auf **Free Aspect**, ganz unten auf das **Plus** (damit fügen wir eine neue Größe hinzu), wählen bei **Type** die **Aspect Ratio** und geben eine Breite und Höhe von **X=9 Y=16** an.



Wenn du die Umrandung der Kamera nicht sehen kannst, drück auf diesen Button rechts oben in der Scene-View, dieser muss blau sein.

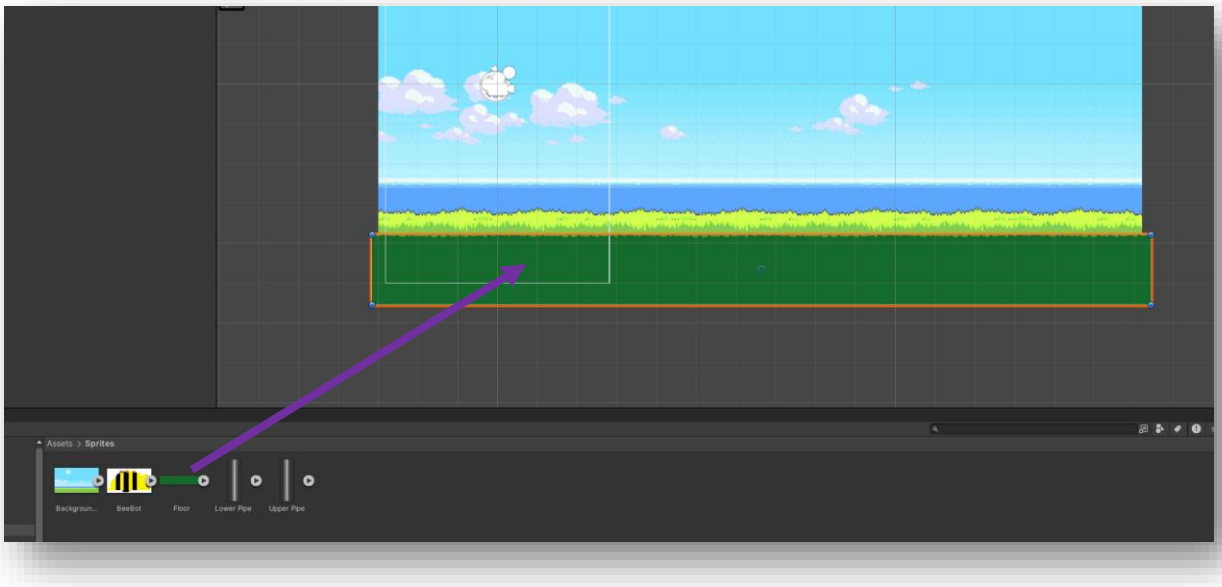


Wechseln wir nun wieder in die Scene zurück (klick auf den Reiter links neben Game) und **ziehen** den **Background** in die Scene. Passe dessen Größe an die Kameraumrandung an und lass das Bild ruhig rechts über die Kamera rausstehen.

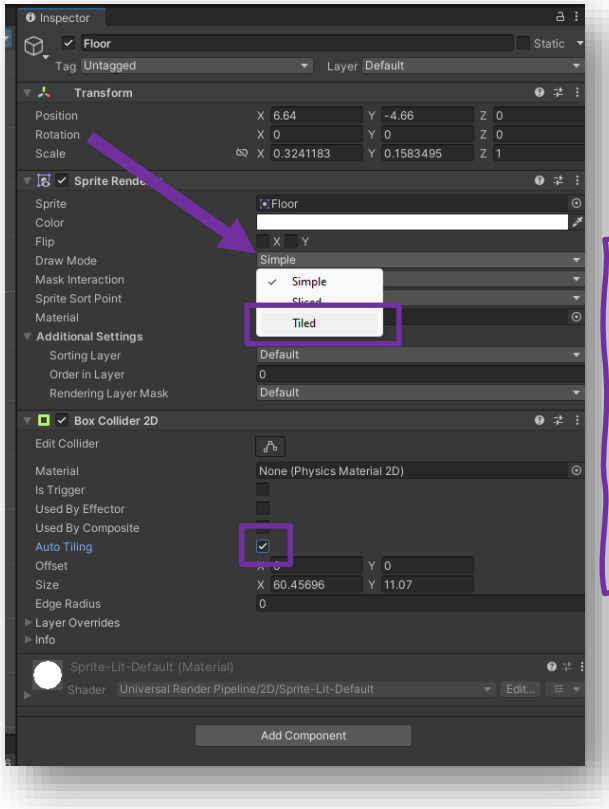


Der Boden (Floor)

Im Anschluss machen wir das gleiche mit **Floor**. Wir ziehen es in unsere Scene und passen die **Größe** so an, wie du es im **Bild unten** siehst. Wenn der Floor hinter dem Hintergrund verschwindet mach dir keine Gedanken, wir ändern das später.



Diesem neuen Object **Floor** geben wir nun die Komponente **Box Collider 2D** (im Inspector unter Add Component → Physics 2D → Box Collider 2D).



Nun ändern wir im Inspector von **Floor** den **Draw Mode** (in Komponente Sprite Renderer) auf **Tiled.**

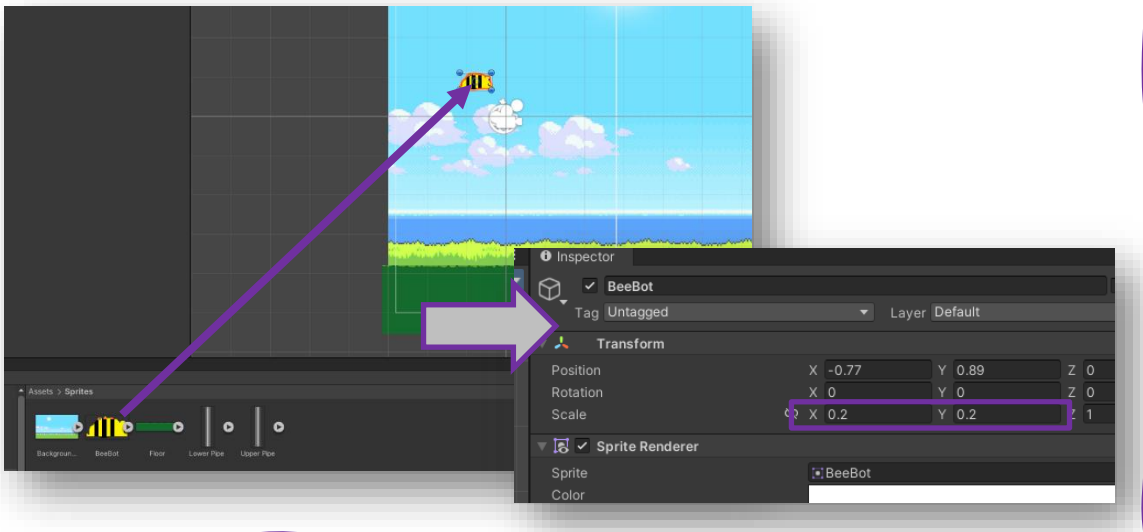
In Komponente Box Collider 2D setzen wir **Auto Tiling** auf **Wahr**.

Dadurch passt sich der Box Collider automatisch an die Größe von Floor an.

Die Biene (BeeBot)

Da wir nun unseren Hintergrund (**Background**) sowie unseren Boden (**Floor**) haben, bringen wir nun unseren Charakter (**BeeBot**) ins Spiel. Wenn die Biene hinter dem Hintergrund verschwindet ist das nicht schlimm, wir ändern das später.

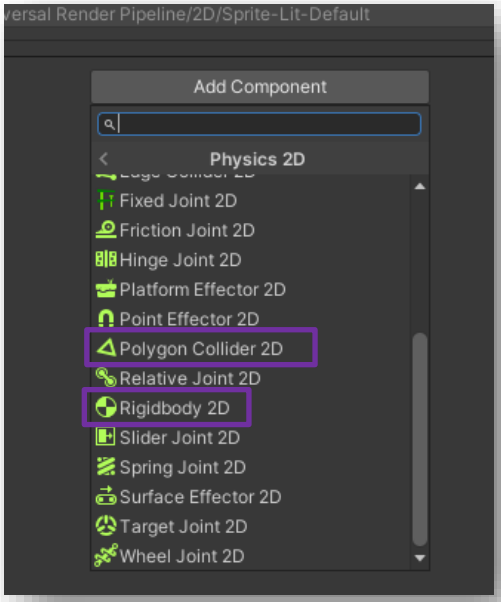
Dazu ziehen wir **BeeBot**, gleich wie den Background und Floor, aus unserem **Sprites Ordner** in die Scene und ändern dessen **Scale** (im Inspector) auf **0.2** (X und Y Wert).



Wir geben dem **BeeBot** nun **2 Komponenten** (im Inspector), nämlich **Polygon Collider 2D** und **Rigidbody 2D**.

Der **Collider** ermöglicht es uns, **Zusammenstöße** mit anderen Objekten wahrzunehmen.

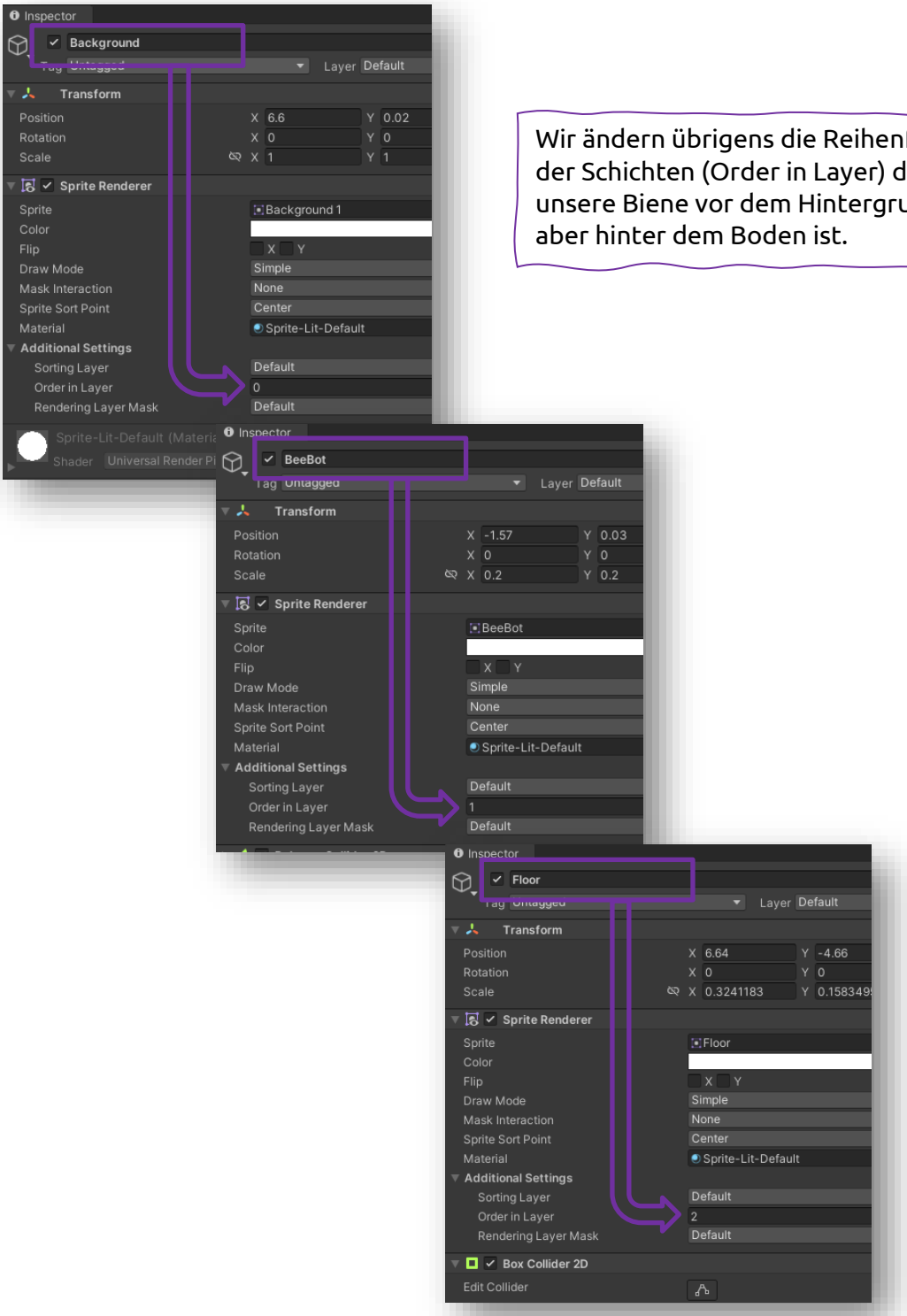
Rigidbody gibt uns die Möglichkeit, **Gravitation** auf unseren BeeBot anzuwenden.



Schichten (Layer)

Wähle nun deinen Hintergrund (**Background**) aus, navigiere zur Komponente **Sprite Renderer** (im Inspector), klicke auf **Additional Settings** (um diese Aufzuklappen) und ändere den Wert von **Order in Layer** auf 0.

Mache nun das gleiche für die Biene (**BeeBot**) aber mit dem **Wert 1** und für den Boden (**Floor**) mit dem **Wert 2**.



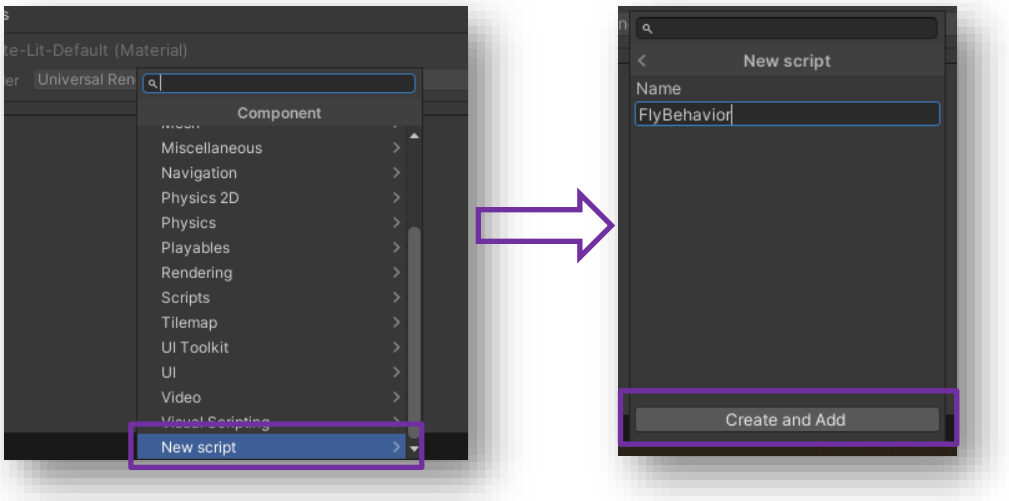
Wir ändern übrigens die Reihenfolge der Schichten (Order in Layer) dass unsere Biene vor dem Hintergrund, aber hinter dem Boden ist.

Skript

Wir haben nun eine **Biene** die hinunterfällt wenn wir das Spiel starten. Nun möchten wir die **Biene steuern** und **fliegen lassen**. Dies können wir, indem wir ein neues Skript bei unserer Biene anlegen.

Wähle dafür die **Biene** aus und klicke (im Inspector) auf „**Add Component → New Script**“ und nenne es „**FlyBehavior**“.

Achte, wie auch bei unserem letzten Spiel, dass du den Skript-Namen auch wirklich richtig geschrieben hast.



Mit einem Doppelklick auf das Skript öffnet sich dieses in unserer Programmierumgebung.

Kopiere nun den **Code** von der **nächsten Seite** in dieses Programm.

(Markiere alles in deinem Skript mit „Strg + A“ und lösche es. Markiere nun alles von Programm auf der nächsten Seite und kopiere es mit „Strg + C“. Mit „Strg + V“ fügst du das dann in dein Skript ein)

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class FlyBehavior : MonoBehaviour
{

    public float velocity = 5.1f;
    float rotationSpeed = 8f;

    Rigidbody2D rigidbody2D;

    void Start()
    {
        rigidbody2D = GetComponent<Rigidbody2D>();
    }

    void Update()
    {
        if (Input.GetMouseButtonDown(0)) {
            rigidbody2D.velocity = Vector2.up * velocity;
        }
    }

    void FixedUpdate() {
        transform.rotation = Quaternion.Euler(0, 0, rigidbody2D.velocity.y * rotationSpeed);
    }
}
```

MZS_FlyBehavior.txt

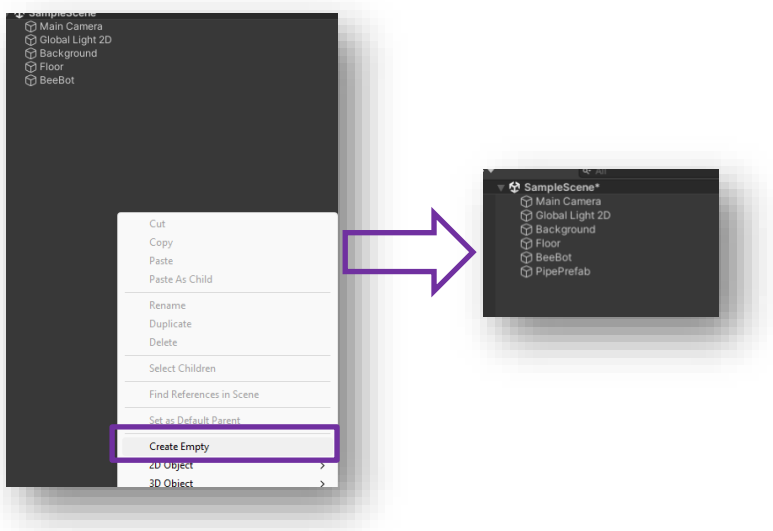
Nachdem du den oberen **Code** in dein **Skript kopiert** hast, drücke auf **Speichern** (oder „Strg + S“ und **wechsle** zurück nach **Unity**. Nach einer kurzen Ladezeit kannst du das Spiel starten.

Hast du alles richtig gemacht, kannst du nun das Spiel Starten (der Play-Button mittig oben in der Scene) und mit einem **linken Mausklick** deine **Biene fliegen** lassen.

Röhren (Pipes)

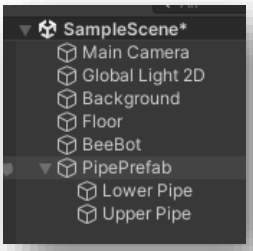
Um die Röhren (**Pipes**), denen unsere **Biene ausweichen** soll, ins Spiel zu bringen, **erstellen** wir in der Hierarchy ein **neues leeres Objekt** (Create Empty).

Dieses nennen wir „**PipePrefab**“.



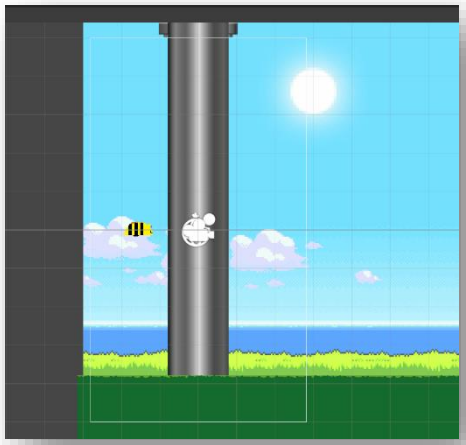
In unser **PipePrefab** Objekt ziehen wir nun aus unserem **Sprites** Ordner (in Project) unsere **Lower Pipe** und unsere **Upper Pipe**.

Das müsste dann in der Hierarchy so aussehen



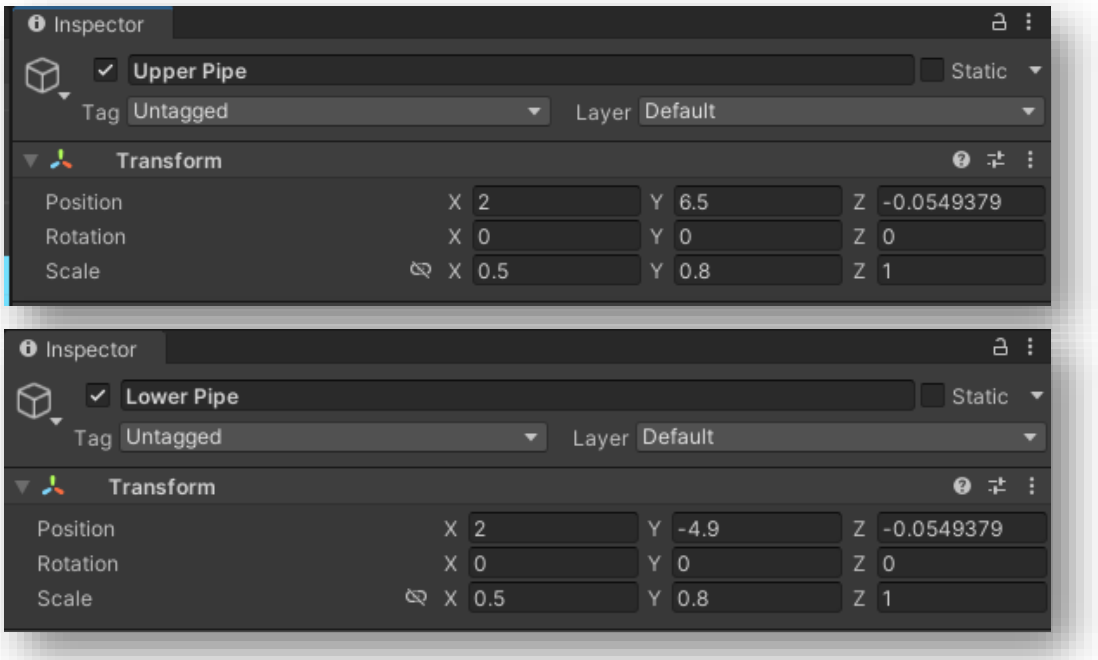
Ändern wir nun die „**Order in Layer**“ von **Lower Pipe** und **Upper Pipe**. Wähle dazu das Objekt aus, gehe in den Inspector und in der Komponente Sprite Renderer klappst du die **Additional Settings** aus.

Ändere nun die **Order in Layer** von 0 auf **1**. Dadurch sollten die Röhren (Pipes) vor den Hintergrund, aber hinter den Boden gesetzt werden.



Sehr gut. Nun passen wir noch die Röhren (Pipes) so an, dass die **Biene** dazwischen durchfliegen kann. Du kannst auch **gerne diese Einstellungen** hier übernehmen. Du siehst im Bild **links oben** welche Röhre welche Einstellungen hat (Upper Pipe und Lower Pipe).

Wähle die jeweilige Röhre (Upper oder Lower Pipe) aus, schau in den Inspector und in der Komponente **Transform** kannst du die Einstellungen anpassen.



Box Collider

Jetzt geben wir unseren **Röhren** noch **Box Collider**, damit das Spiel auch registriert, wenn wir in eine der Röhren fliegen.

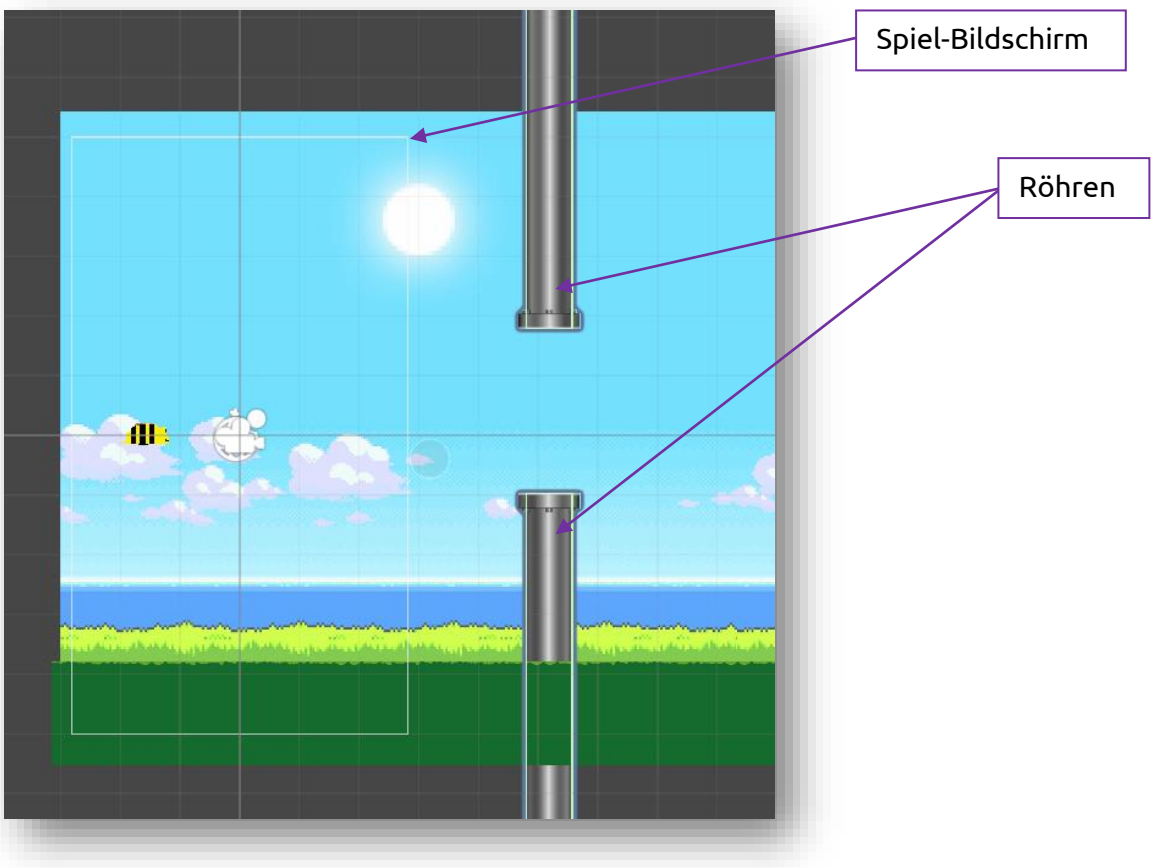
Wir wählen wieder die jeweilige Röhre aus, klicken auf „**Add Component**“ (im Inspector) und unter **Physics2D** wählen wir **BoxCollider2D** aus.

Vergiss nicht den Box Collider bei beiden Röhren hinzuzufügen.

Passe nun noch die Größe (**Size**) beider **Box Collider** an, dass das **grüne Rechteck** komplett in die Röhre passt.

In diesem Fall wurde der Wert von **X** auf **1.55** geändert.

Nun, da wir die Röhren ins Spiel gebracht haben und diese mit passenden „**Collidern**“ ausgestattet haben, wählen wir unser Objekt **PipePrefab** (in der Hierarchy) aus. Im Inspector siehst du nun die Position des Objekts. Verändere bei **Position** den **X-Wert** so, dass die Röhren **außerhalb** des **Spiel-Bildschirms** sind:



Bewegen der Röhren

Nun erstellen wir für unser **PipPrefab** Objekt ein neues Skript, dieses nennen wir „**MovePipes**“. Also, **PipePrefab** in der Hierarchie auswählen, im Inspector „**Add Component**“ → **new Script** nennen es „**MovePipes“** und auf **Create and Add** klicken.

Achte auch hier wieder darauf, dass du es richtig benennst.

Mache nun das Skript auf, **lösche alles** was darin ist und **kopiere den Code** darunter **in dein Skript** (MovePipes).

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class MovePipes : MonoBehaviour
{

    public float speed = 1.5f;

    // Start is called before the first frame update
    void Start()
    {

    }

    // Update is called once per frame
    void Update()
    {
        transform.position += Vector3.left * speed * Time.deltaTime;
    }
}
```

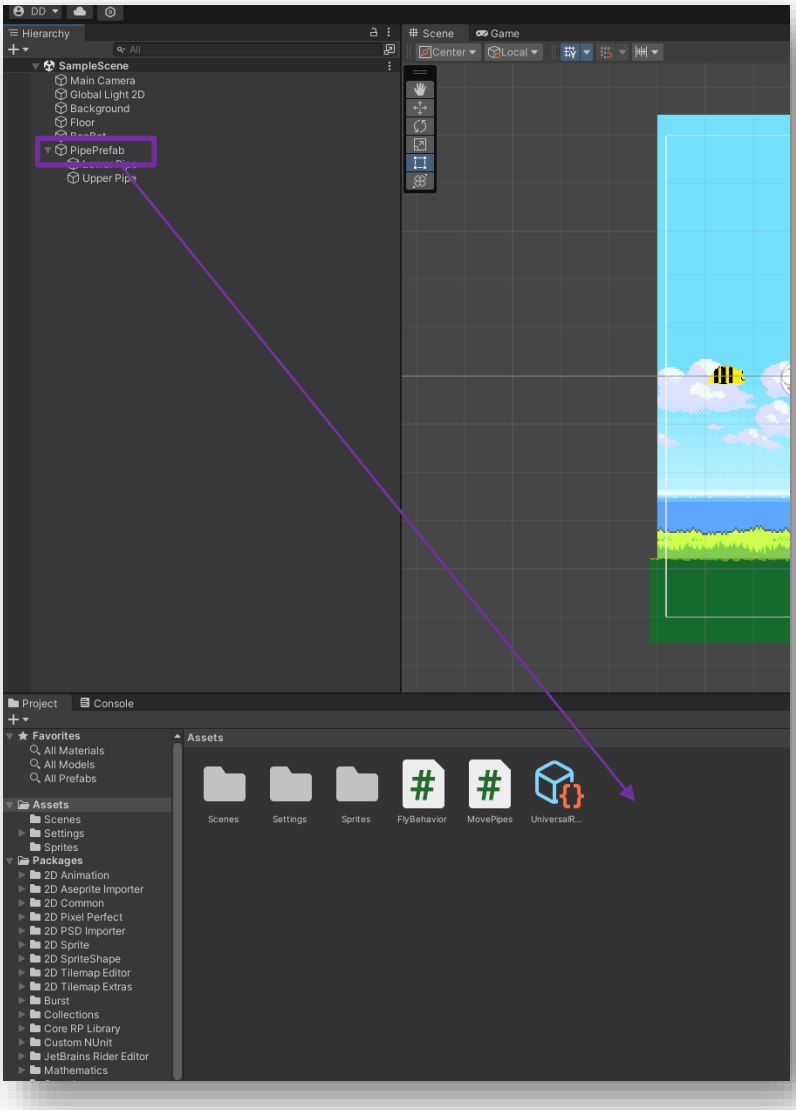
MZS_MovePipes.txt

Speichere das Skript nun und **wechsle** zurück zu **Unity**. Wenn du das Spiel jetzt startest, müssten sich deine Röhren nach links bewegen.

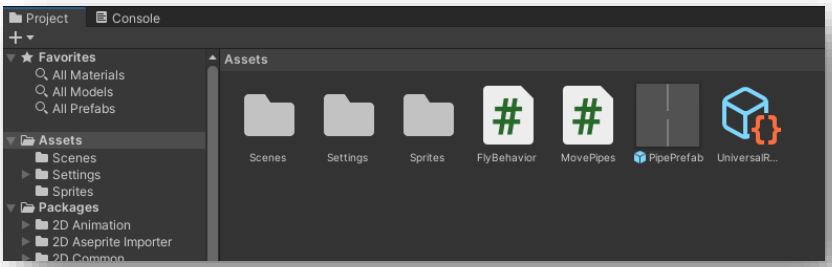
Außerdem darf die Biene nicht durch eine der Röhren hindurch fliegen (natürlich schon durch das Loch zwischen den Röhren, nicht aber durch eine Röhre selbst). Sollte sie das trotzdem tun, überprüfe nochmal ob du die **Collider** bei den Objekten richtig gesetzt hast.

Röhren Spawnen lassen

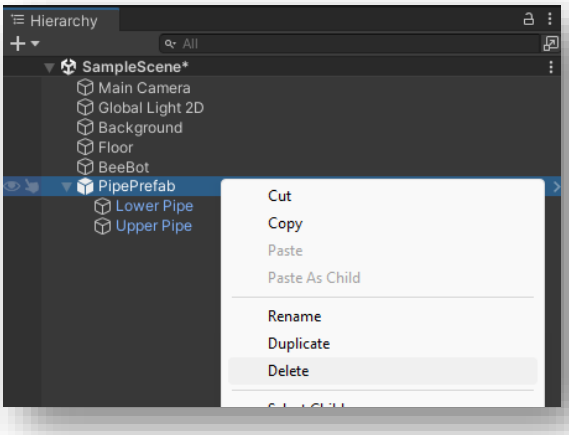
Da sich die Röhren jetzt bewegen brauchen wir natürlich mehr als nur ein Hindernis. Wir möchten, dass die Röhren dauernd spawnen. Hierfür ziehen wir nun unser **PipePrefab** Objekt aus der Hierarchy nach **Assets** (in Project).



Dein **Assets** Ordner sollte nun so aussehen:



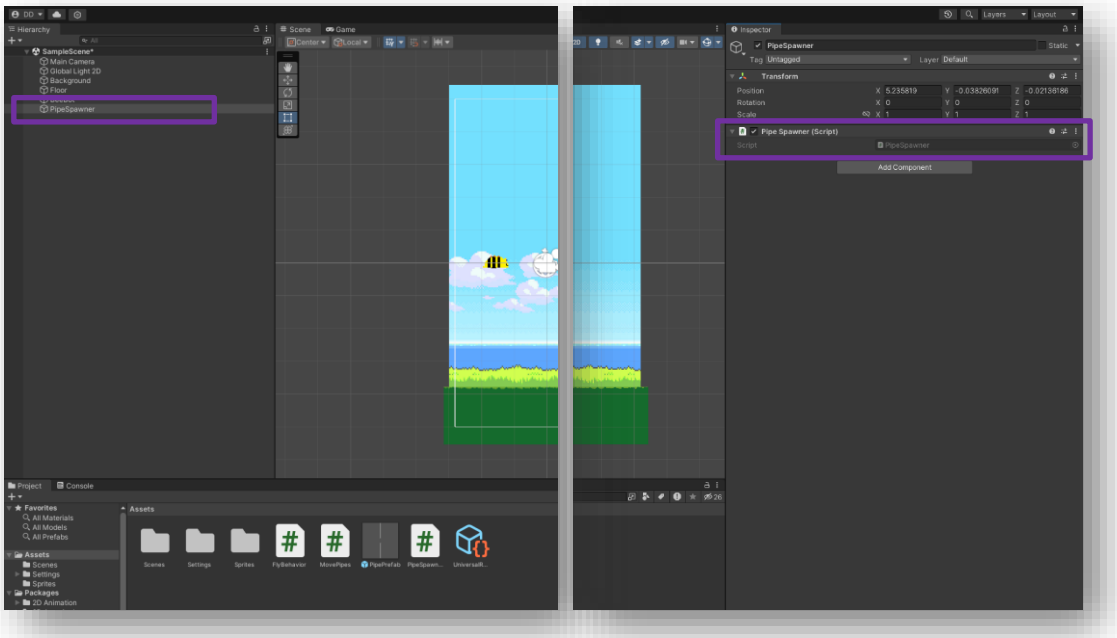
Jetzt löschen wir das Objekt **PipePrefab** aus unserer Hierarchy (**Rechtsklick** und dann **Delete**).



Als Nächstes erstellen wir in der Hierarchy ein **Create Empty** und nennen es „**PipeSpawner**“.

Diesem Objekt geben wir nun ein **neues Skript**, dieses nennen wir ebenfalls „**PipeSpawner**“.

Das sollte dann so aussehen:



In dieses **Script kopierst** du nun den **Code** von der **nächsten Seite**:

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;


public class PipeSpawner : MonoBehaviour
{
    public float delay = 3.0f;
    public GameObject pipes;


    // Start is called before the first frame update
    void Start()
    {
        InvokeRepeating("Spawn", 0, delay);
    }


    // Update is called once per frame
    void Update()
    {


    }


    void Spawn() {
        Destroy(Instantiate(pipes, new Vector3(3, Random.Range(-2.5f,2.5f), 0), Quaternion.identity),6);
    }
}
```

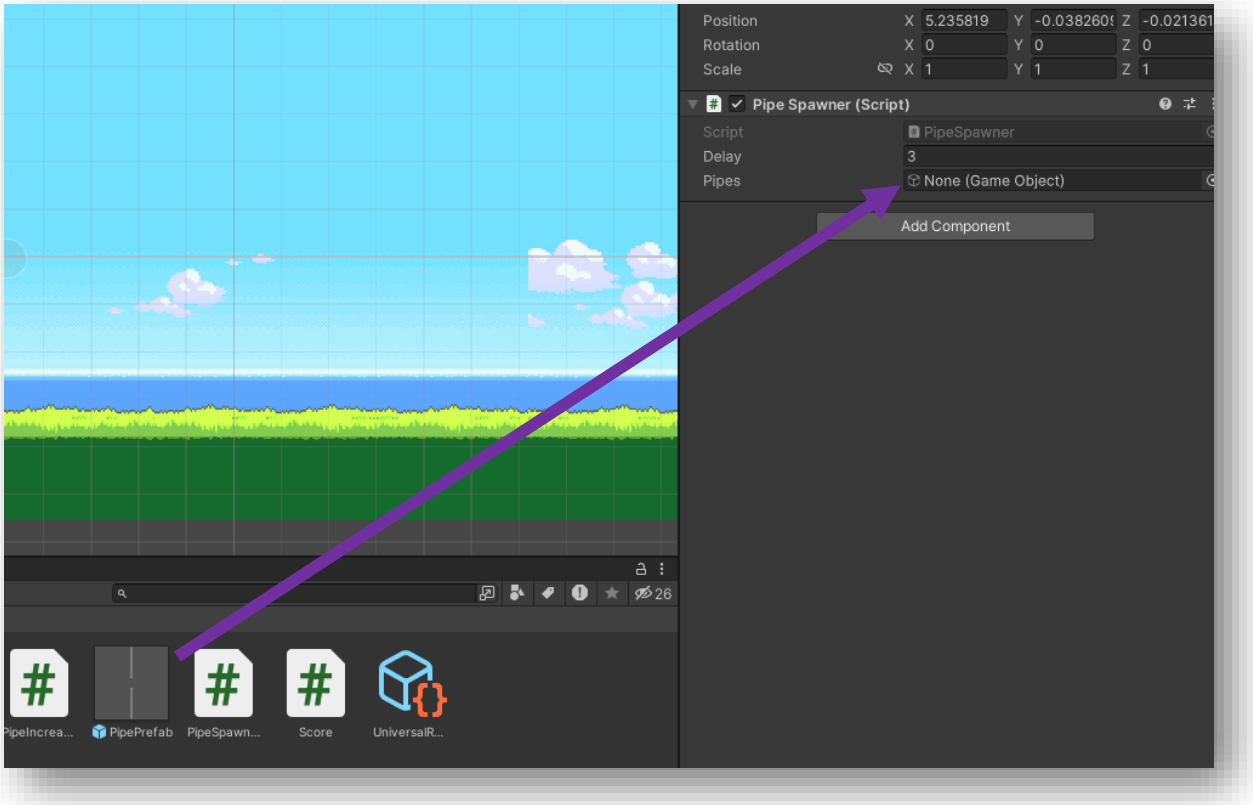
MZS_PipeSpawner.txt

Objekte referenzieren

Nun müssen wir noch unsere PipePrefab mit unserem Skript verknüpfen.

Ziehe dafür das **PipePrefab** Objekt aus Project (im Ordner **Assets**) in die noch leere „**Pipes**“ **Variable** von unsrem **PipeSpawner**-Script (im Inspector).

(Vergiss nicht, das **PipeSpawner** Objekt zuerst in der Hierarchy auszuwählen dass du das Skript überhaupt im Inspector siehst).

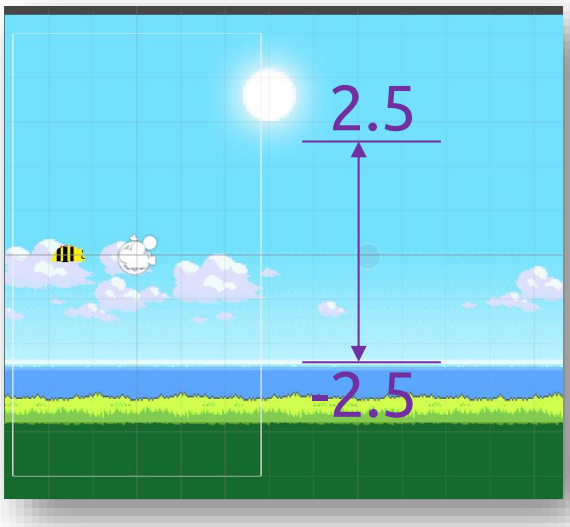


Bitte beachte, dass du den Code gegebenenfalls anpassen musst. Es kann nämlich sein, dass die Position der gespawnten Röhre für dich nicht ganz so passt. Ändere dafür diese beiden Zahlen:

Destroy(Instantiate(pipes, new Vector3(3, Random.Range(-2.5f,2.5f), 0), Quaternion.identity),6);

Das sind die **Zahlen 2.5** und **-2.5** in **Float** (das ist ein Zahlentyp).

In unserem Fall bedeutet das, dass die Röhren **zufällig** in **Y-Richtung** zwischen **-2.5** und **2.5 spawnen können**. Zufällig übrigens wegen „Random.Range(…)“:



Nun können wir die Biene fliegen lassen, die Röhren spawnen von selbst und sie bewegen sich.

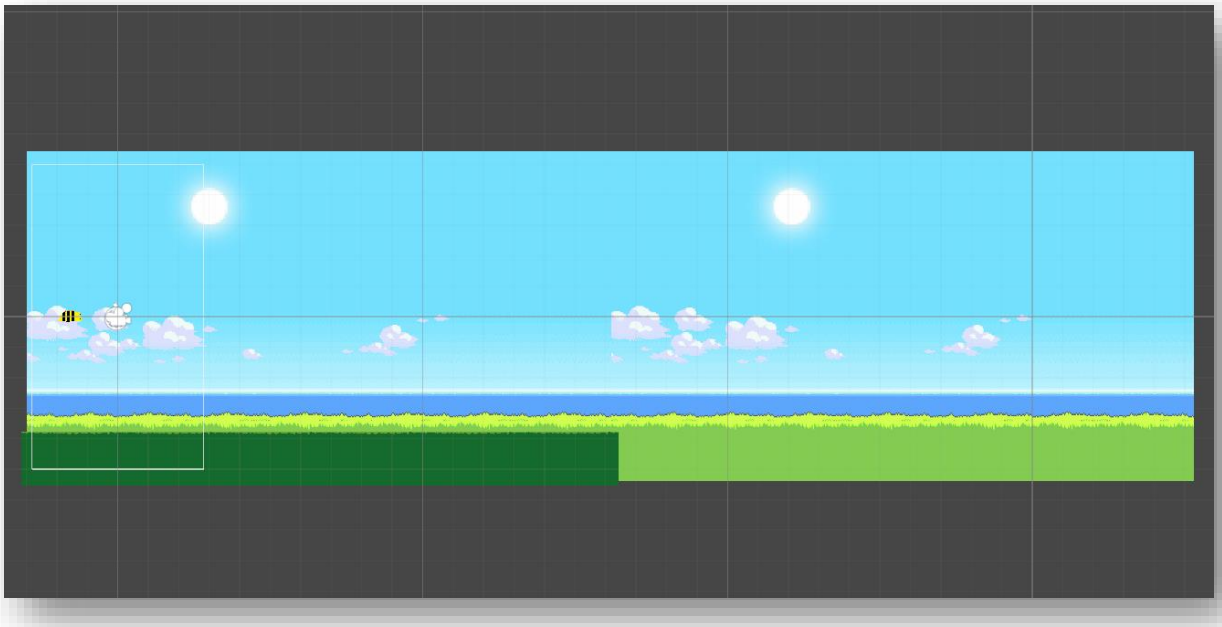
Nun möchten wir noch, dass sich der Hintergrund in Spielrichtung, also nach links, bewegt.

Den Hintergrund bewegen

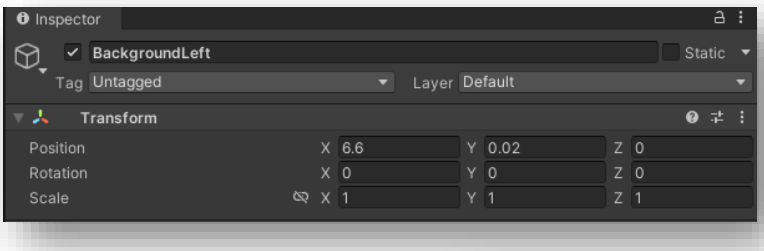
Kopiere nun den **Hintergrund** und **verschiebe** ihn so weit nach **rechts (X-Achse),** dass du einen schönen Übergang zwischen den beiden Hintergründen hast. Wenn die Wolken im Hintergrund abgeschnitten sind macht das überhaupt nichts.

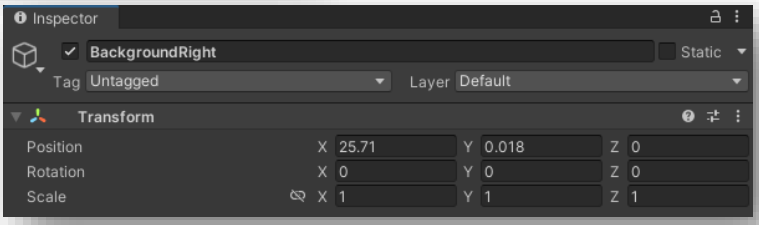
Benenne den linken Hintergrund „**BackgroundLeft**“ und den rechten „**BackgroundRight**“.

Es sollte dann so aussehen:



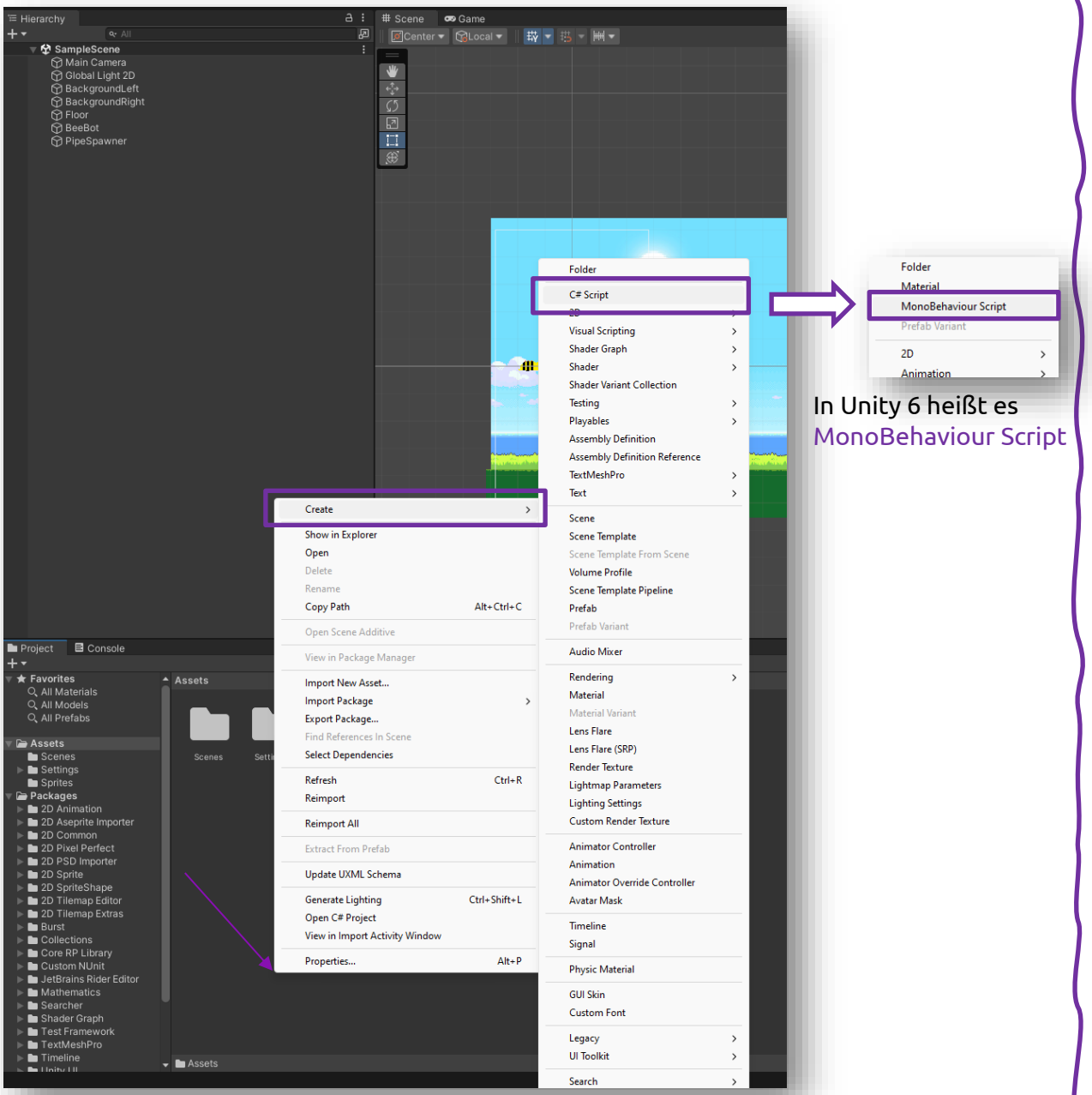
Hier sind die Werte für die beiden Hintergründe vom oberen Bild, du kannst sie gerne für dein Projekt übernehmen





Jetzt erstellen wir ein **neues Skript**, um den Hintergrund zu bewegen. Wie man ein Skript erstellt, weißt du als Profi natürlich schon. Erstelle das Skript diesmal aber in Project.

Also im **Assets**-Ordner in Project rechtsklicken, **Create → C# Script** (bzw. **MonoBehaviour Script** – So heißt es in Unity 6)

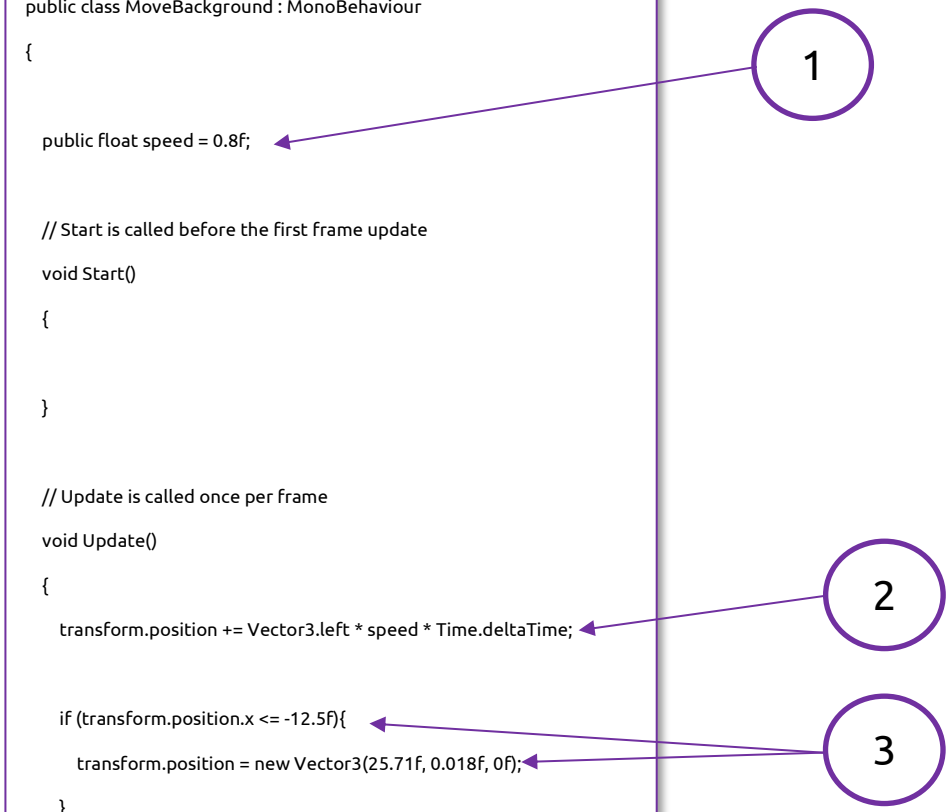


In Unity 6 heißt es MonoBehaviour Script

Dieses Skript nennen wir „**MoveBackground**“. Achte wieder darauf, dass du es richtig geschrieben hast.

Kopiere diesen **Code** in dein **Script**:

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class MoveBackground : MonoBehaviour
{

  public float speed = 0.8f;

  // Start is called before the first frame update
  void Start()
  {

  }

  // Update is called once per frame
  void Update()
  {
    transform.position += Vector3.left * speed * Time.deltaTime;

    if (transform.position.x <= -12.5f){
      transform.position = new Vector3(25.71f, 0.018f, 0f);
    }
  }
}
```



MZS_MoveBackground.txt

1: Die **Geschwindigkeitsvariable:** Sie gibt an, wie schnell sich der Hintergrund bewegt

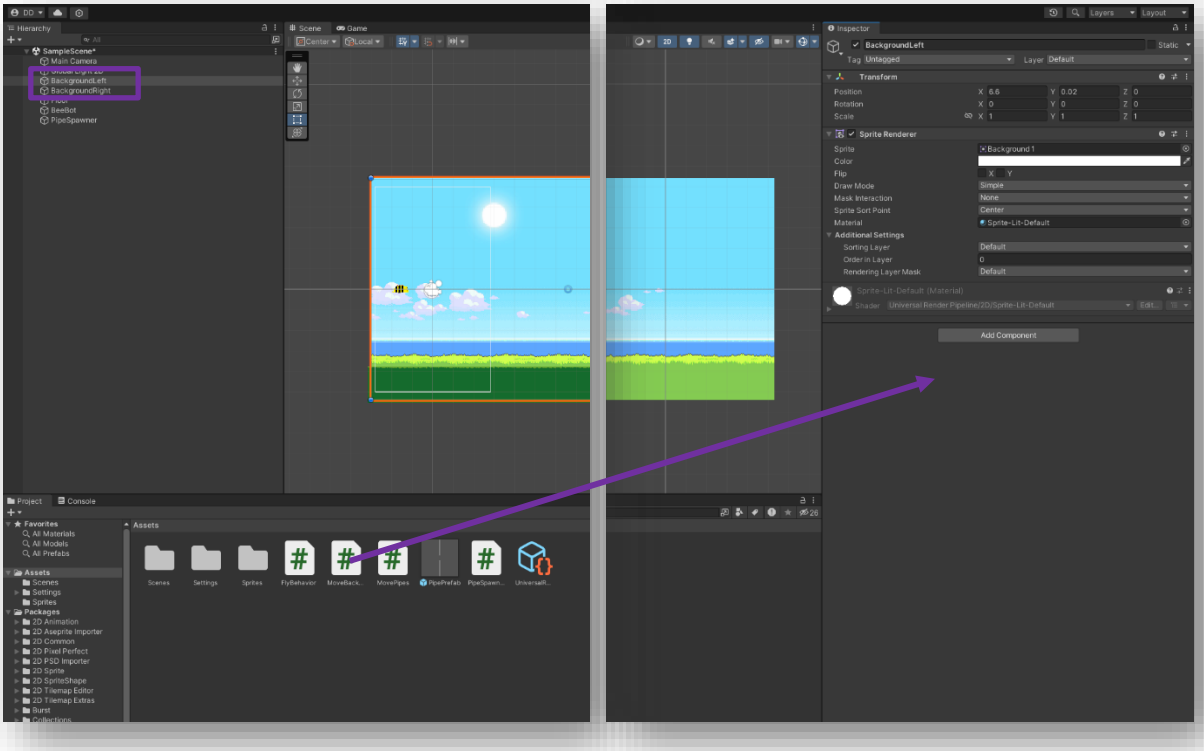
2: Hier wird der **Hintergrund** nach **links verschoben**

3: Wenn der **Hintergrund** einen gleich bzw. kleineren **X-Wert als 12.5** hat, wird der **Hintergrund** nach (X) **25.71**, (Y) **0.018** und (Z) **0 verschoben**.

Nun haben wir das fertige Skript und müssen es nur noch bei unseren beiden Hintergründen einbinden.

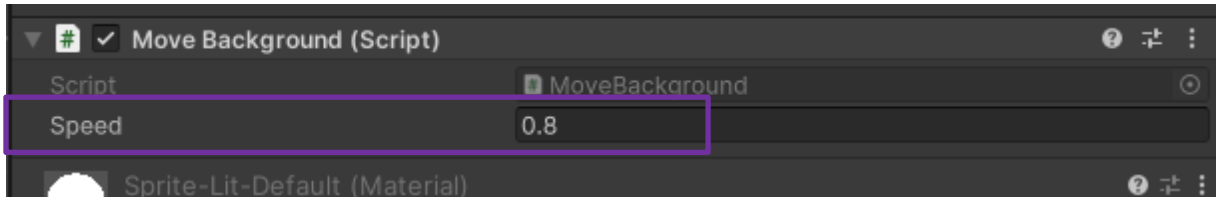
Klicke dafür auf jeweils einen der Hintergründe (**BackgroundLeft** oder **BackgroundRight**) in der Hierarchy und ziehe unser **MoveBackground**-Skript von Project in den Inspector unter die anderen **Komponenten**.

Du musst dies für beide Hintergründe einzeln machen.



Sehr gut. Wenn du jetzt das Spiel startest, müssen sich die Hintergründe nach links bewegen.

Wenn du die **Geschwindigkeit** des Hintergrunds **anpassen willst**, musst du nur die Variable „**speed**“ im **MoveBackground**-Skript ändern. Du kannst auch im Inspector, während das Spiel läuft die Variable **Speed** ändern.

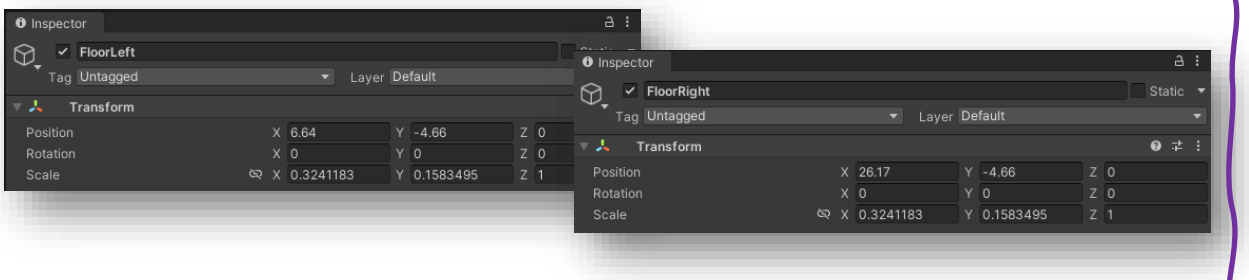


Den Boden bewegen

Wenn wir nun den auch den Boden bewegen möchten, müssen wir genau das Gleiche wie beim Hintergrund machen. Folge, falls du dir nicht ganz sicher bist, folge einfach der Anleitung vom „**Den Hintergrund bewegen**“.

Hier sind nochmal die einzelnen Schritte:

1. **Kopiere** den **Boden** und **verschiebe** ihn entlang der **X-Achse** so weit nach **rechts**, dass wir einen nahtlosen Übergang haben.
2. **Benenne** nun den linken Boden **FloorLeft** und den rechten **FloorRight**.
   (Hier sind die Eigenschaften von beiden)



3. Erstelle im **Assets**-Ordner in Project ein **C#Script** und nenne es „**MoveFloor**“.
4. **Kopiere** den **Code** der nächsten Seite in dieses Skript.
5. Ziehe das **Script** unter die Komponenten (im Inspector) von **FloorLeft** und **FloorRight.**

Code für **MoveFloor**-Skript:

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class MoveFloor : MonoBehaviour
{

    public float speed = 1.5f;

    // Start is called before the first frame update
    void Start()
    {

    }

    // Update is called once per frame
    void Update()
    {
        transform.position += Vector3.left * speed * Time.deltaTime;

        if (transform.position.x <= -12.8f){
            transform.position = new Vector3(26.17f, -4.66f, 0f);
        }
    }
}
```

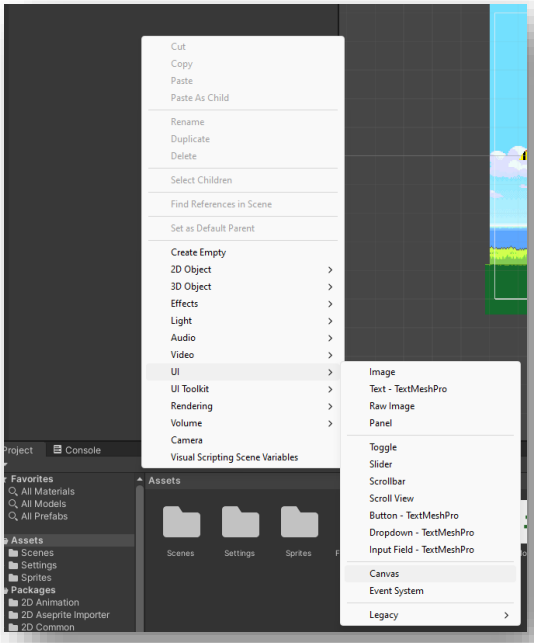
MZS_MoveFloor.txt

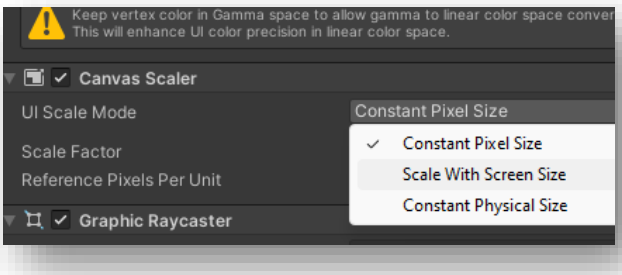
Game Over Screen

Nun erstellen wir unseren “Game Over Screen“. Er soll erscheinen, wenn unsere Biene den Boden, oder in eine der Röhren berührt.

Wir erstellen zunächst in der Hierarchy ein **Canvas**. Also, Rechtsklick in der Hierarchy, **UI → Canvas** und nenne es „**GameOverScreen**“.

Anmerkung: Wenn wir ein Canvas erstellen, wird automatisch ein Objekt namens EventSystem mitangelegt.

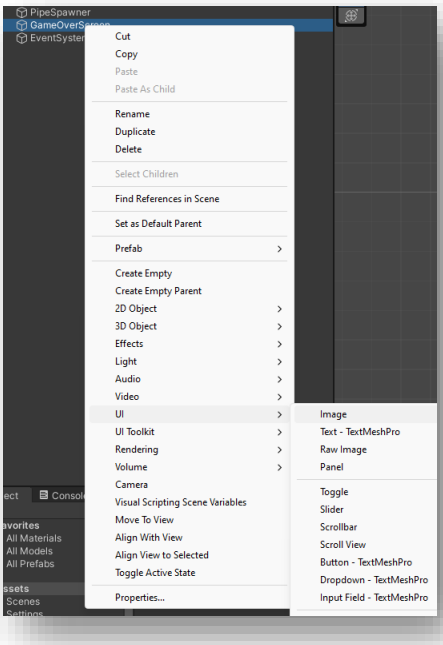




In der Komponente Canvas Scaler (im Inspector) ändern wir den Wert von **UI Scale Mode** von *Constant Pixel Size* zu **Scale With Screen Size**.

Nun fügen wir 2 Bilder in unseren **GameOverScreen** ein.

Also, Rechtsklick auf **GameOverScreen**, **UI → Image**. Das ganze 2 mal.

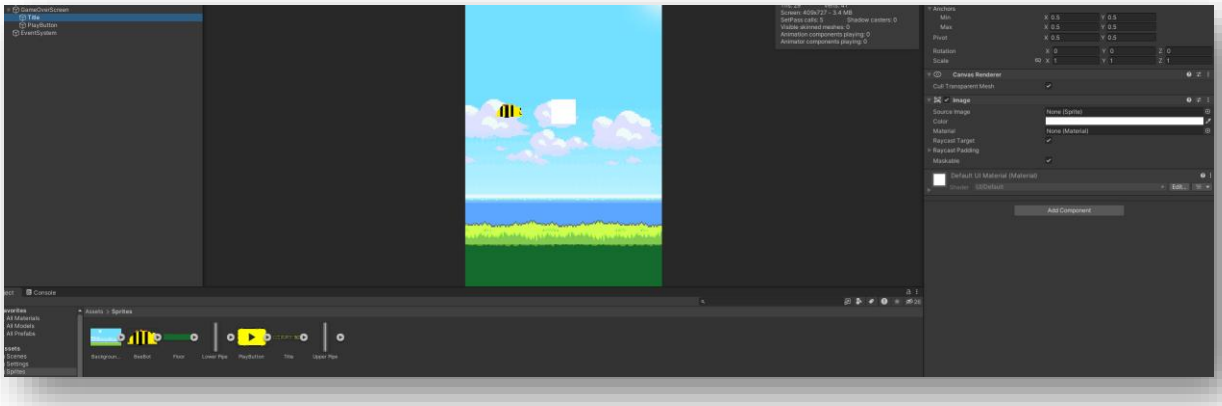


Die zwei Bilder (Images) nennen wir:

1. **Title**
2. **PlayButton**

Wechsle nun für die nächsten Schritte in die Spielansicht wenn du in der Scene bist. Über dem **Spielbildschirm** links oben auf „**Game**“ wechseln. In der Scene-Ansicht siehst du nämlich die Objekte die wir bearbeiten nicht.

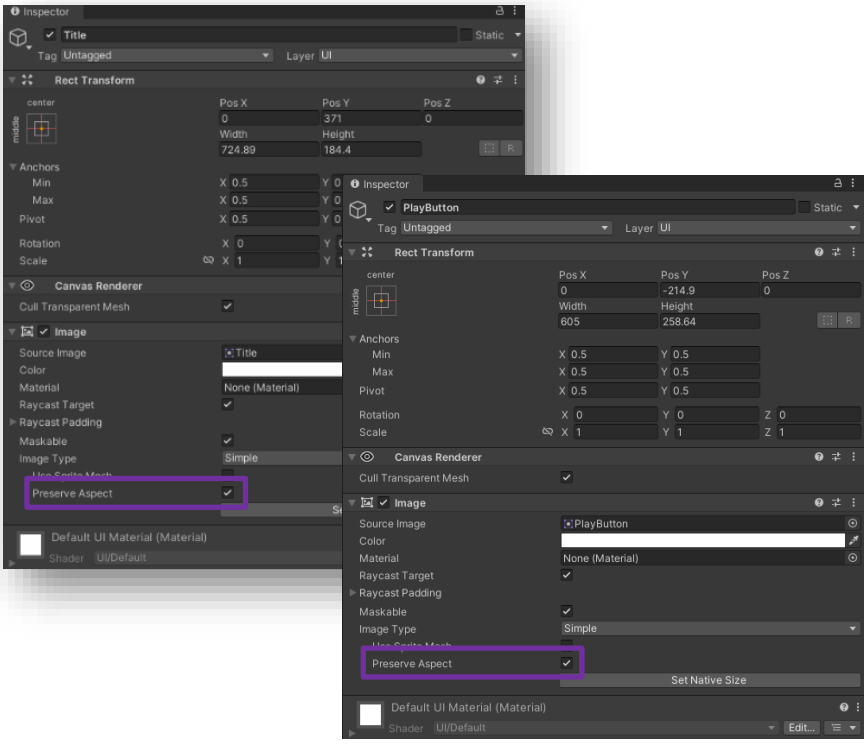
Wähle nun unseren **Title** aus (das Objekt unter GameOverScreen) und ziehe unseren **Title**.png (das Bild) in **Source Image** hinein (unter Image im Inspector).

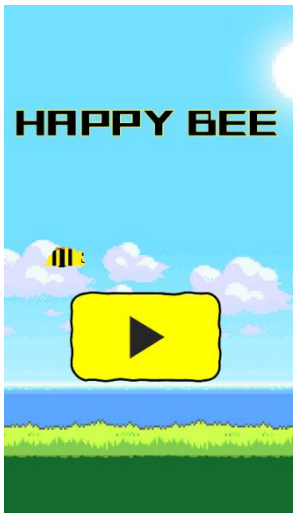


Mach nun das gleiche mit dem **PlayButton-**Objekt und dem **PlayButton**.png-Bild

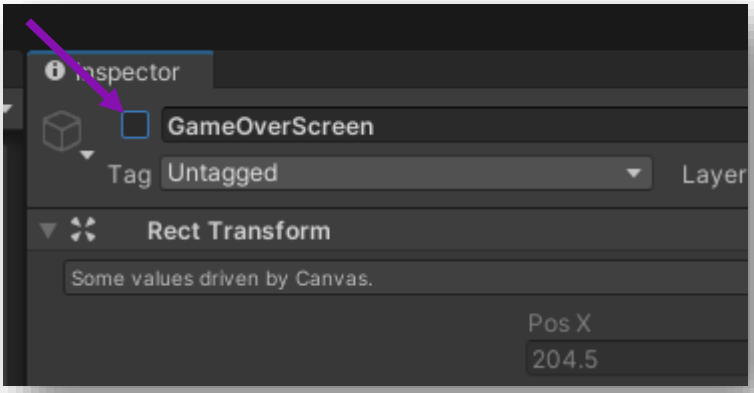
Mache die folgenden Punkte für **Title** und **PlayButton**:

1. Wähle das **Objekt** in der Hierarchy aus
2. Im Inspector klicke auf **Set Native Size** (in der Komponente **Image**)
3. Setze das Häkchen bei **Preserve Aspect**
4. Ändere nun in der Komponente Rect Transform deine **Pos X** und **Pos Y**, sowie **Width** und **Height** so, dass du damit zufrieden bist (Alternativ kannst du diese Werte verwenden).





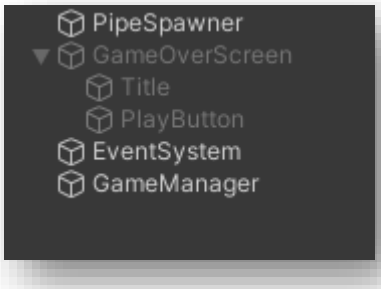
Nun **deaktivieren** wir unseren **GameOverScreen** indem wir das Häkchen links vom Namen im Inspector entfernen.



Title und PlayButton sollten nun nichtmehr im Spiel sichtbar sein.

Der GameManager

Nun erzeugen wir noch ein neues Objekt in der Hierarchy (**Create Empty**) Namens „**GameManager**“.



Diesem Objekt geben wir nun noch **ein neues Skript**, dieses nennen wir „**GameManager**“.

Wähle das Objekt in der Hierarchy aus, und fügen im Inspector ein neues Skript hinzu. Also: **Add Component → New Script → „GameManager“ → Create and Add**

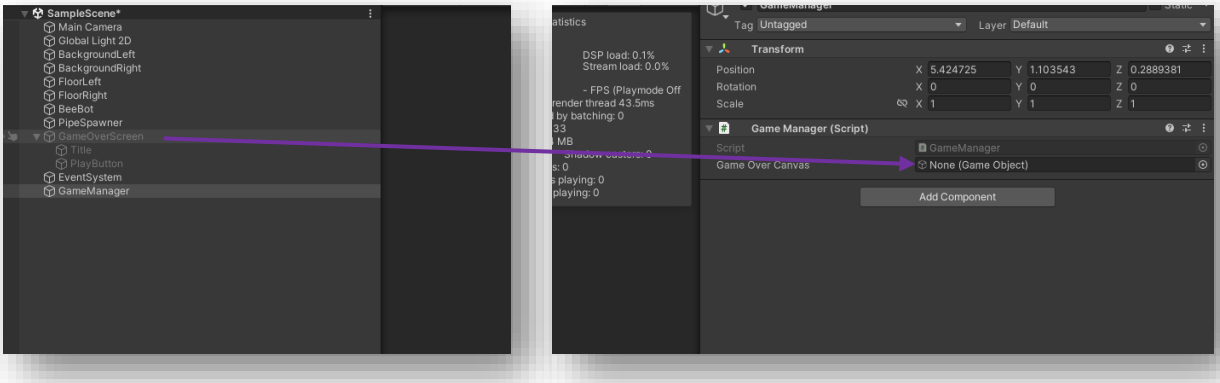
Kopiere in dieses Skript nun den Code von der nächsten Seite (vergiss nicht alles zu markieren, sodass wirklich nur der Code der nächsten Seite in dem Skript steht)

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;

public class GameManager : MonoBehaviour
{
    public static GameManager instance;

    public GameObject gameOverCanvas;

    void Awake()
    {
        if (instance == null) {
            instance = this;
        }

        Time.timeScale = 1f;
    }

    public void GameOver() {
        gameOverCanvas.SetActive(true);

        Time.timeScale = 0f;
    }

    public void RestartGame() {
        SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex);
    }
}
```

MZS_GameManager.txt

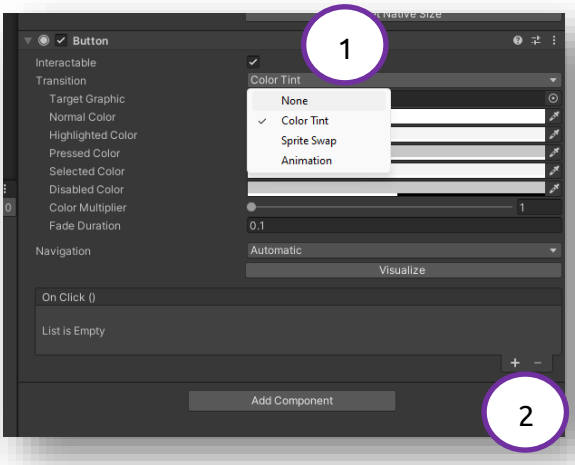
Speichere nun das Skript und wechsle zurück nach Unity.

Wähle in der Hierarchy unseren GameManager an, nun sollte im Inspector unter deinem Skript die Variable **Game Over Canvas** stehen. Wir ziehen nun unseren **GameOverScreen** von der Hierarchy in dieses Feld um es zu referenzieren.

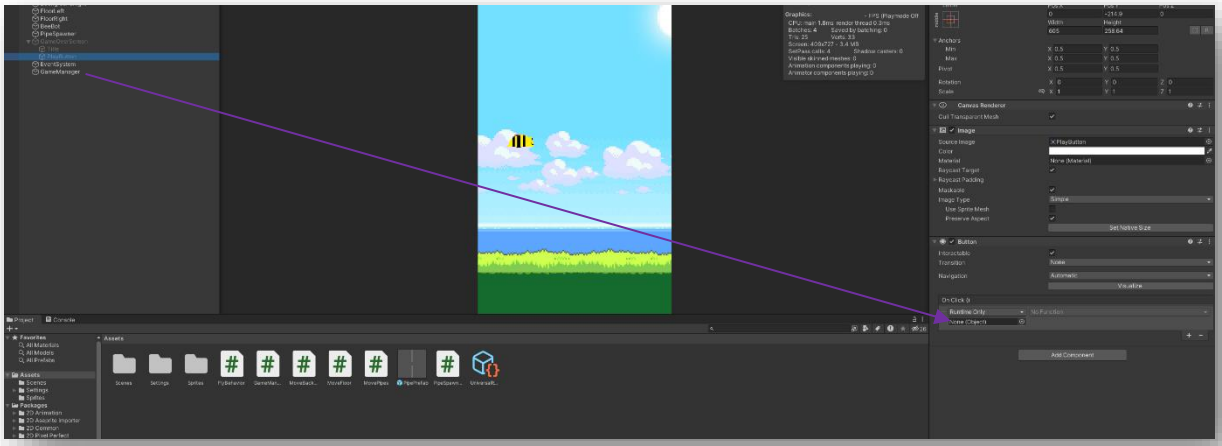


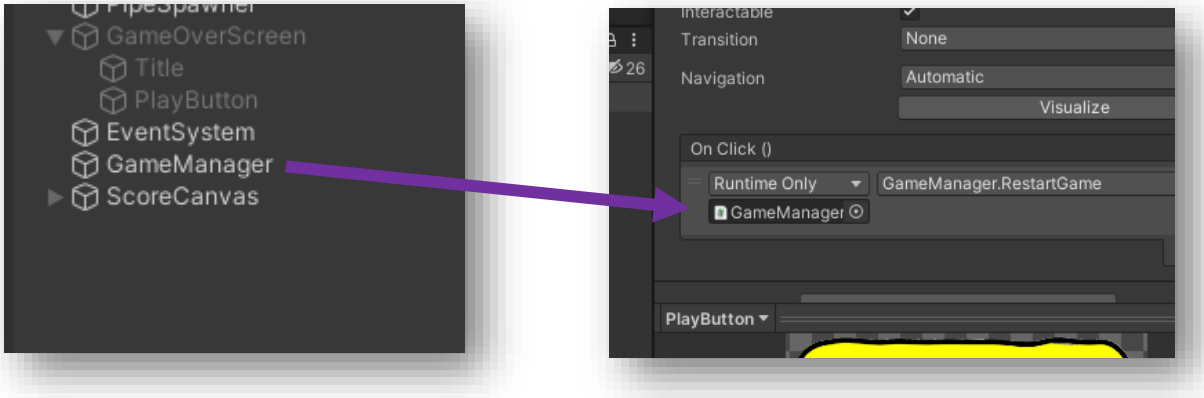
Jetzt geben wir unserem **PlayButton** Objekt eine neue Komponente **Button**. Diese finden wir im Inspector unter **Add Component → UI → Button**

1. Anschließend ändern wir **Transition** zu **None.**
2. Danach klicke auf das „+“ bei On Click () (unten).

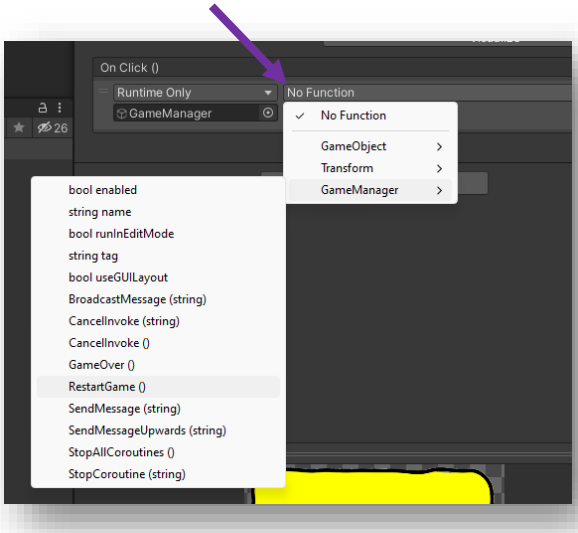


Nun möchten wir in **On Click** (vom Button) das **GameManager**-Objekt referenzieren, also ziehen wir, nachdem wir auf das Plus (+) gedrückt haben, den **GameManager** von der Hierarchy in das Feld (wo jetzt noch **None (Object)** steht) im Inspector.





Klicke nun auf das Feld **No Function**,
dann auf
GameManager → RestartGame ()



Anpassen von Code (FlyBehavior)

Alles, was wir jetzt noch machen müssen ist unser **FlyBehavior** Skript anzupassen.

Öffne dafür das Skript und kopiere den folgenden Code nach der geschwungenen geschlossenen Klammer von „**FixedUpdate**“ hinein.

```
void OnCollisionEnter2D(Collision2D collision){

    GameManager.instance.GameOver();

}
```

Das überarbeitete **FlyBehavior** Skript sollte nun so aussehen:

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class FlyBehavior : MonoBehaviour
{

    public float velocity = 5.1f;
    float rotationSpeed = 8f;

    Rigidbody2D rigidbody2D;

    void Start()
    {
        rigidbody2D = GetComponent<Rigidbody2D>();
    }

    void Update()
    {
        if (Input.GetMouseButtonDown(0)) {
            rigidbody2D.velocity = Vector2.up * velocity;
        }
    }

    void FixedUpdate() {
        transform.rotation = Quaternion.Euler(0, 0, rigidbody2D.velocity.y * rotationSpeed);
    }

    void OnCollisionEnter2D(Collision2D collision){
        GameManager.instance.GameOver();
    }
}
```

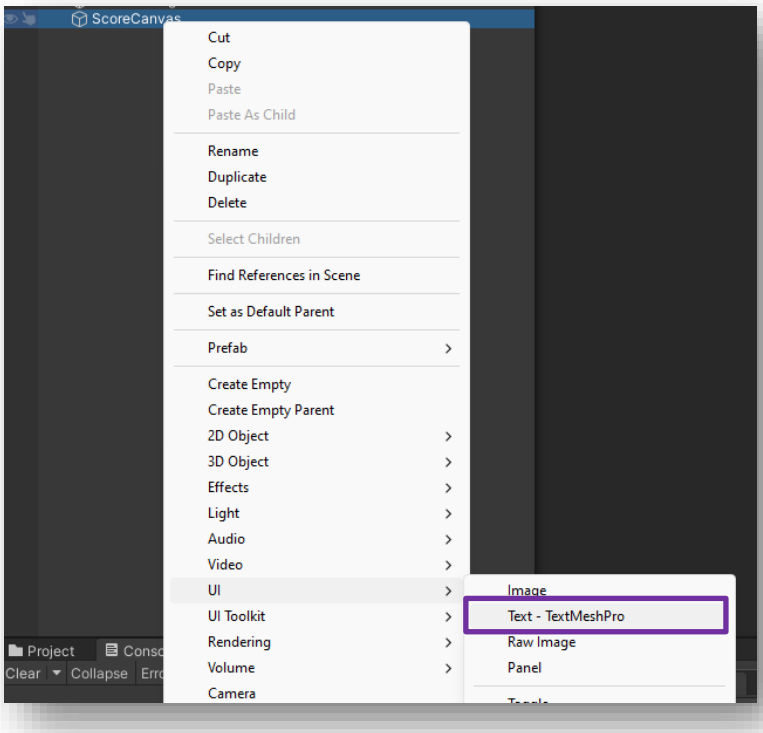
MZS_FlyBehaviorUpdated.txt

Sehr gut! Der letzte Schritt auf unserem Weg besteht nun darin, ein Punktesystem in das Spiel einzuführen.

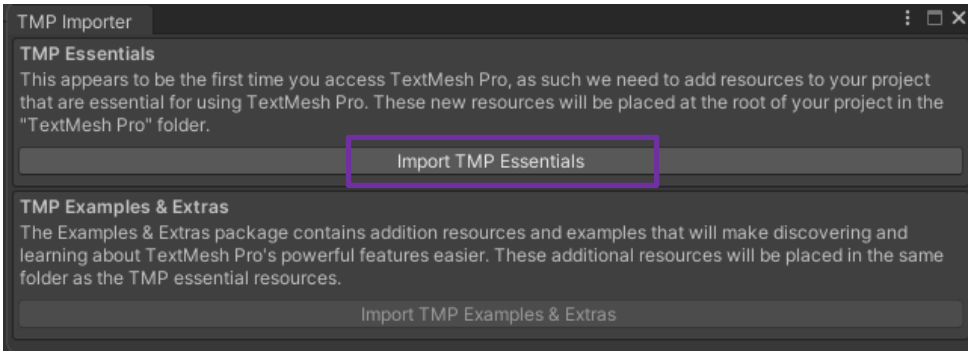
Scoring System

Wir erstellen nun in unserer Hierarchy ein neues **Canvas** (rechtsklicken → UI → Canvas) und nennen es „**ScoreCanvas**“.

In diesem Canvas erzeugen wir anschließend **4 Text - TextMeshPro** Objekte.

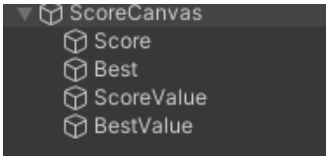


Wenn du gefragt wirst, ob du TMP Essentials importieren möchtest, klicke auf „**Import TMP Essentials**“

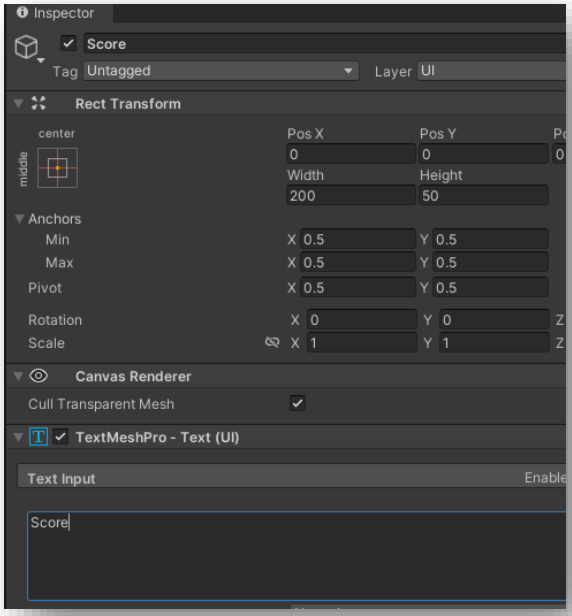


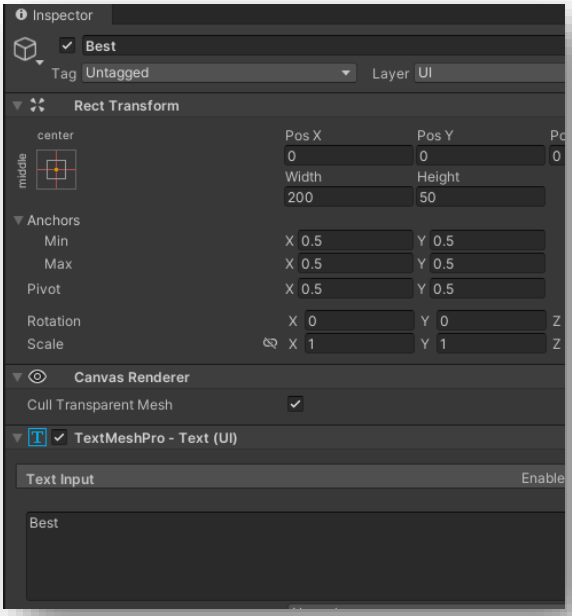
Wir nennen nun unsere 4 TMP (TextMeshPro) Objekte:

1. **Score**
2. **Best**
3. **ScoreValue**
4. **BestValue**



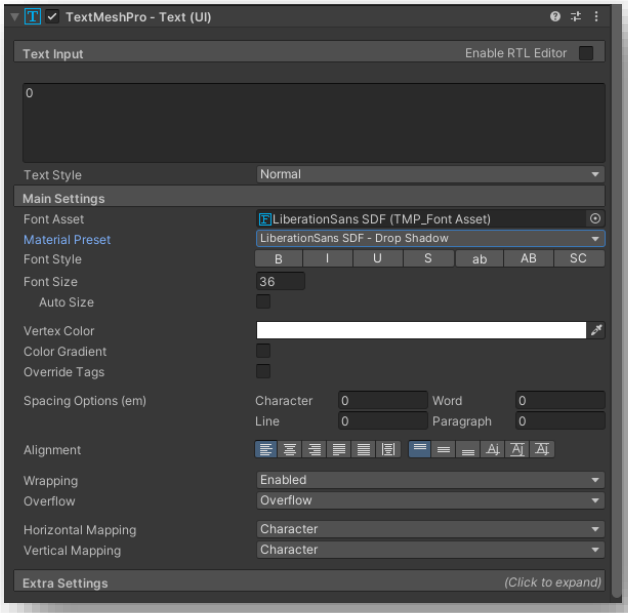
Schreibe auch gleich in **Score** und **Best** jeweils „**Score**“ und „**Best**“ in das Text Input Feld im Inspector:





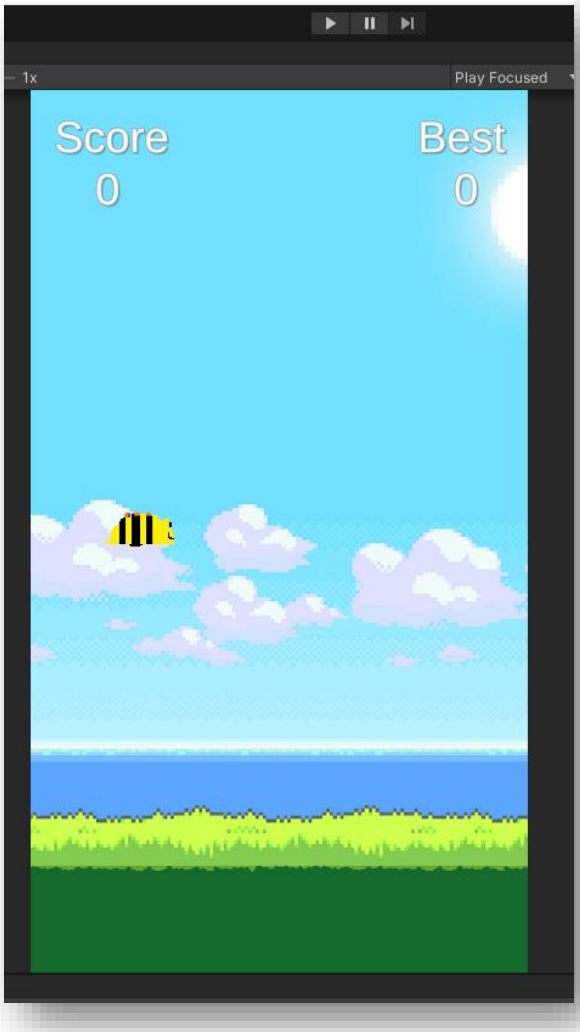
Mache nun das gleiche für **ScoreValue** und **BestValue**, aber schreibe nicht deren Namen in das Input-Feld, sondern die Zahl „**0**“.

Du kannst nun deine Schrift im Inspector nach Belieben einstellen, in diesem Beispiel wurde **Material Preset** auf „LiberationSans SDF – Drop Shadow“ geändert um die Schrift leserlicher zu gestalten.



Richte nun die Textfelder so am Bildschirm aus, wie es dir gefällt. Es könnte zum Beispiel so aussehen:

Achte darauf, dass ScoreValue unter Score ist und BestValue unter Value, dass die richtigen Zahlen dargestellt werden.



Jetzt erstellen wir für unser **ScoreCanvas** ein neues **Script**, dieses nennen „**Score**“.

(**Add Component** im Inspector und **NewScript**)

Öffne es, lösche alles, was darinsteht und kopiere den Code der nächsten Seite hinein.

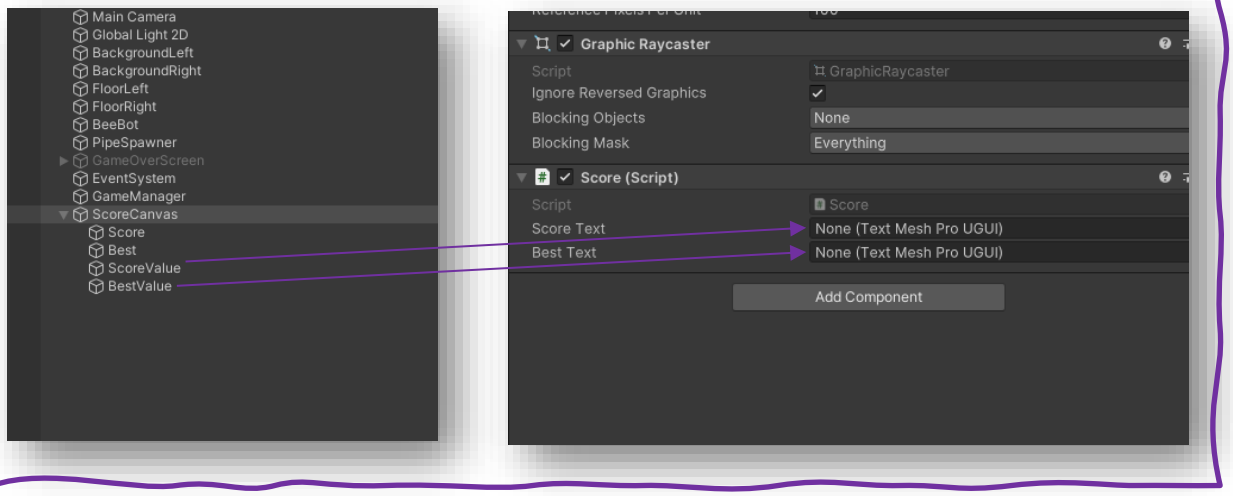
```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using TMPro;

public class Score : MonoBehaviour
{

    public static Score instance;
    public TextMeshProUGUI scoreText;
    public TextMeshProUGUI bestText;
    private int score;

        private void Awake(){
        if (instance == null) {
            instance = this;
        }
    }

    void Start()
    {
        scoreText.text = score.ToString();

        bestText.text = PlayerPrefs.GetInt("HighScore", 0).ToString();
        UpdateBest();
    }

    void UpdateBest()
    {
        if(score > PlayerPrefs.GetInt("HighScore")){
            PlayerPrefs.SetInt("HighScore", score);
            bestText.text = score.ToString();
        }
    }

    public void UpdateScore()
    {
        score++;
        scoreText.text = score.ToString();
        UpdateBest();
    }
}
```

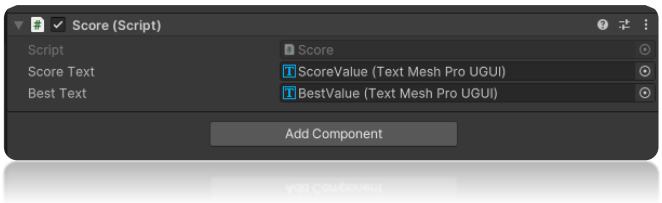
MZS_Score.txt

Nun referenzieren wir unsere **ScoreValue** und **BestValue** Objekte mit dem Skript. Klicke dafür in der Hierarchy auf **ScoreCanvas**.

In dessen **Skript** (im Inspector) solltest du nun die zwei Variablen **Score Text** und **Best Text** sehen.

Ziehe nun das **ScoreValue** Objekt in **Score Text** und **BestValue** Objekt in **Best Text** hinein.

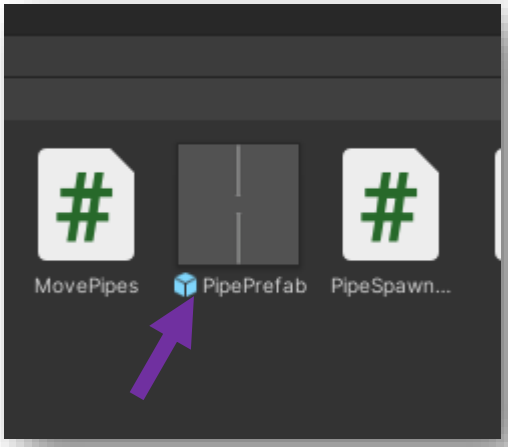




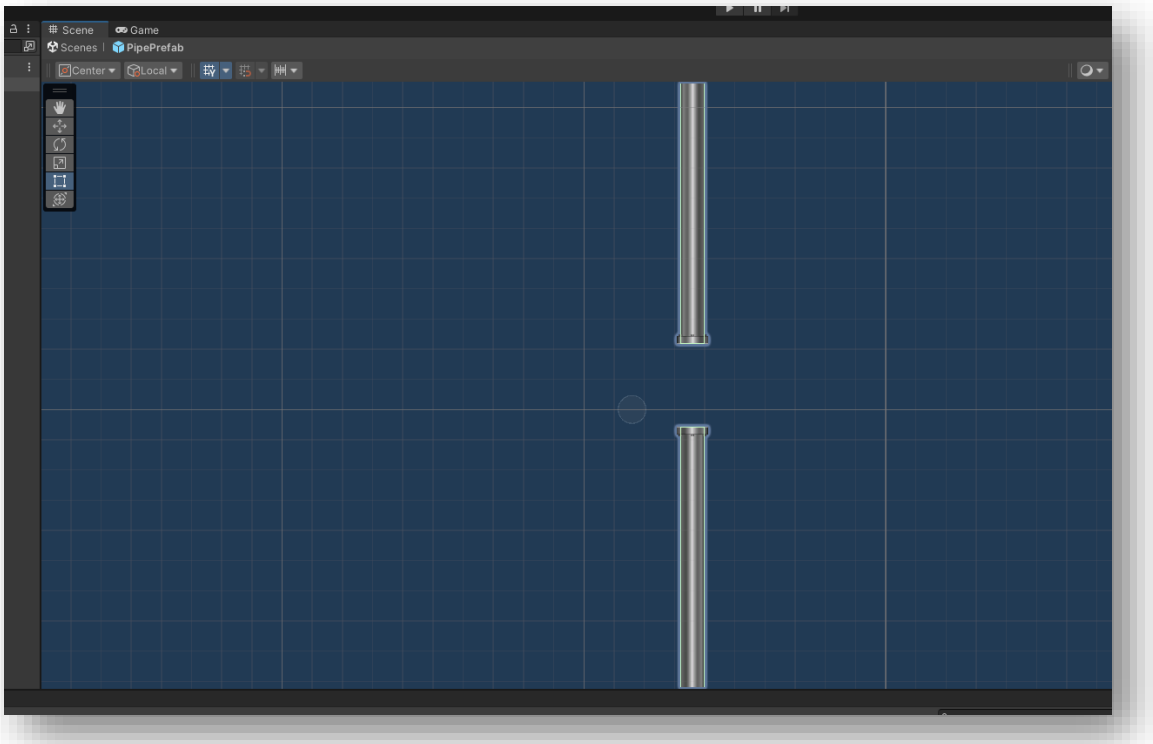
Punkte erhöhen - BoxCollider Trigger

Da wir unsere **Punktezahl** (Score) erhöhen möchten, wenn wir durch die **Röhren** (Pipes) fliegen, brauchen wir noch ein **Skript**, dass uns dies ermöglicht.

Dafür **doppelklicken** wir in Project auf unser „**PipePrefab**“ Objekt.

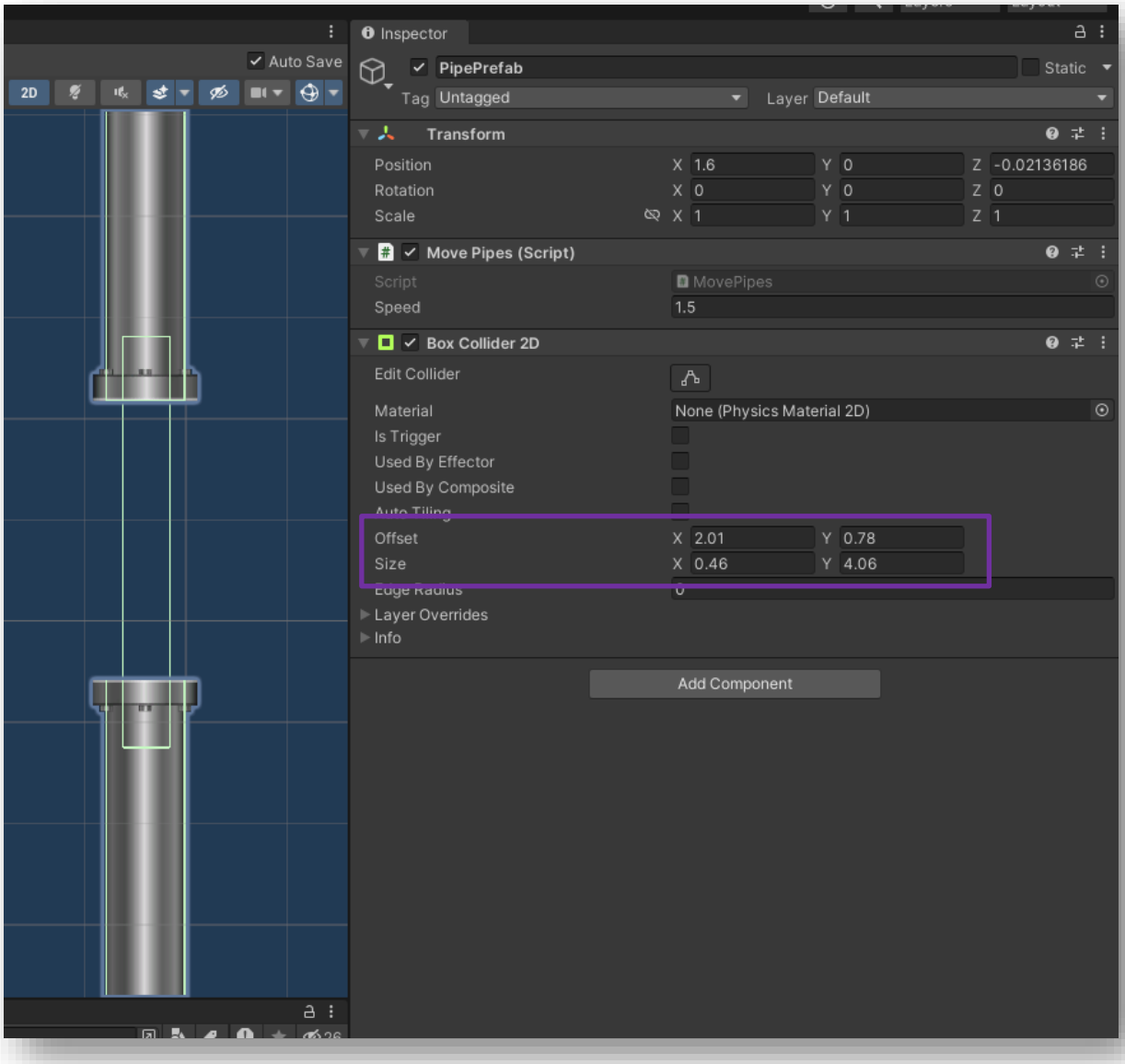


Wechsle nun gegebenenfalls von **Game** auf **Scene** und du solltest das sehen:



Nun fügen wir im Inspector einen **BoxCollider** als **Komponente** zu unserem **PipePrefab** hinzu und platzieren ihn zwischen unsere Röhren
Add Component → Physics 2D → Box Collider 2D

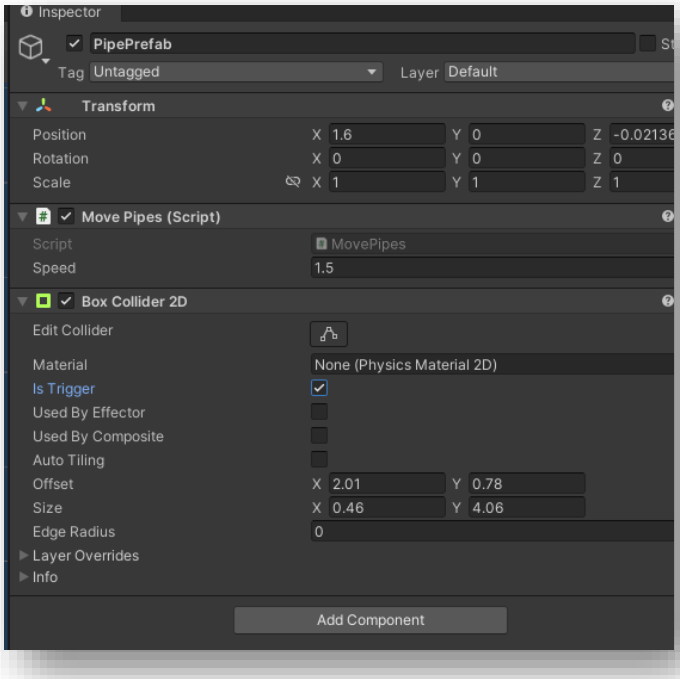
Verändere die **Position** des Box Colliders **mit Offset** und die **Größe mit Size**.



Wozu brauchen wir diesen Box Collider überhaupt?

Er ist dazu da, **um zu registrieren**, ob unsere Biene durch die Röhren fliegt.

Aber um die Biene **nicht zu stoppen,** setzen wir noch das **Häckchen** bei **Is Trigger.**



Anschließend fügen wir zu unserem **PipePrefab** noch ein **Skript** hinzu, um die Punkte zu erhöhen wenn unsere **Biene durchfliegt**.

Dieses **NewScript** nennen wir „**PipeIncreaseScore**“.

Lösche nun alles darin und kopiere den Code darunter in dieses Script:

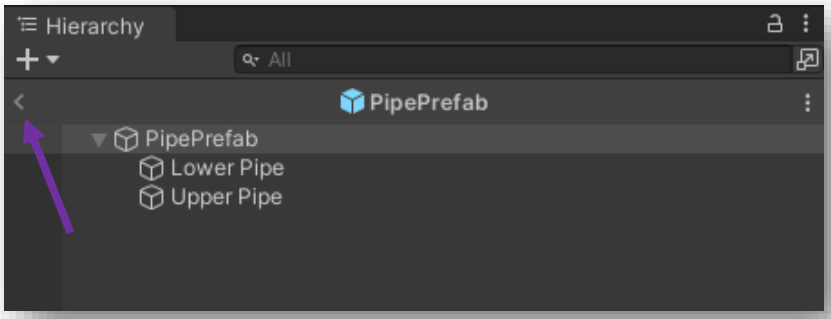
```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;


public class PipeIncreaseScore : MonoBehaviour
{


    private void OnTriggerEnter2D(Collider2D collision) {
        if(collision.gameObject.CompareTag("Player")) {
            Score.instance.UpdateScore();
        }
    }
}
```

MZS_PipeIncreaseScore.txt

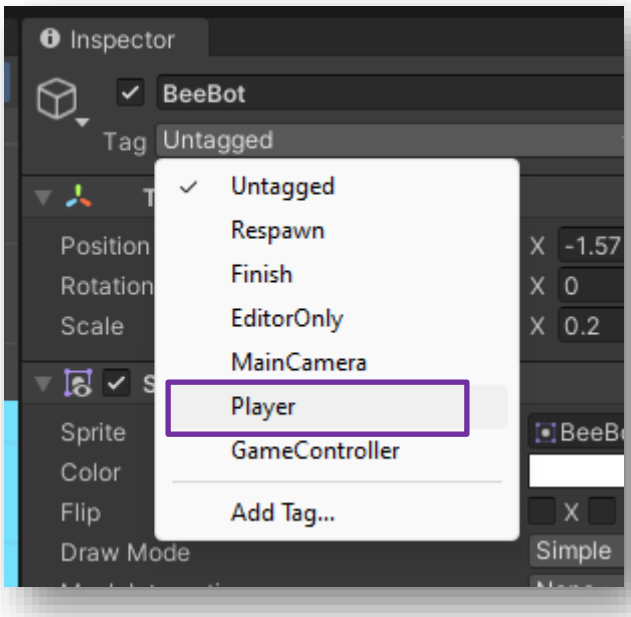
Tags (BeeBot)

Nun wählen wir unsere **Biene** (**BeeBot**) aus und geben ihr im Inspector den **Tag** „**Player**“.

Hier kommst du zurück zu unserer normalen Hierarchy-Ansicht:



Also wählen wir in der Hierarchy **BeeBot** aus und klicken im Inspector bei **Tag** auf „**Player**“.



Und das wars auch schon!

Herzlich Glückwunsch, du hast nun dein zweites 2D Spiel in Unity geschaffen.