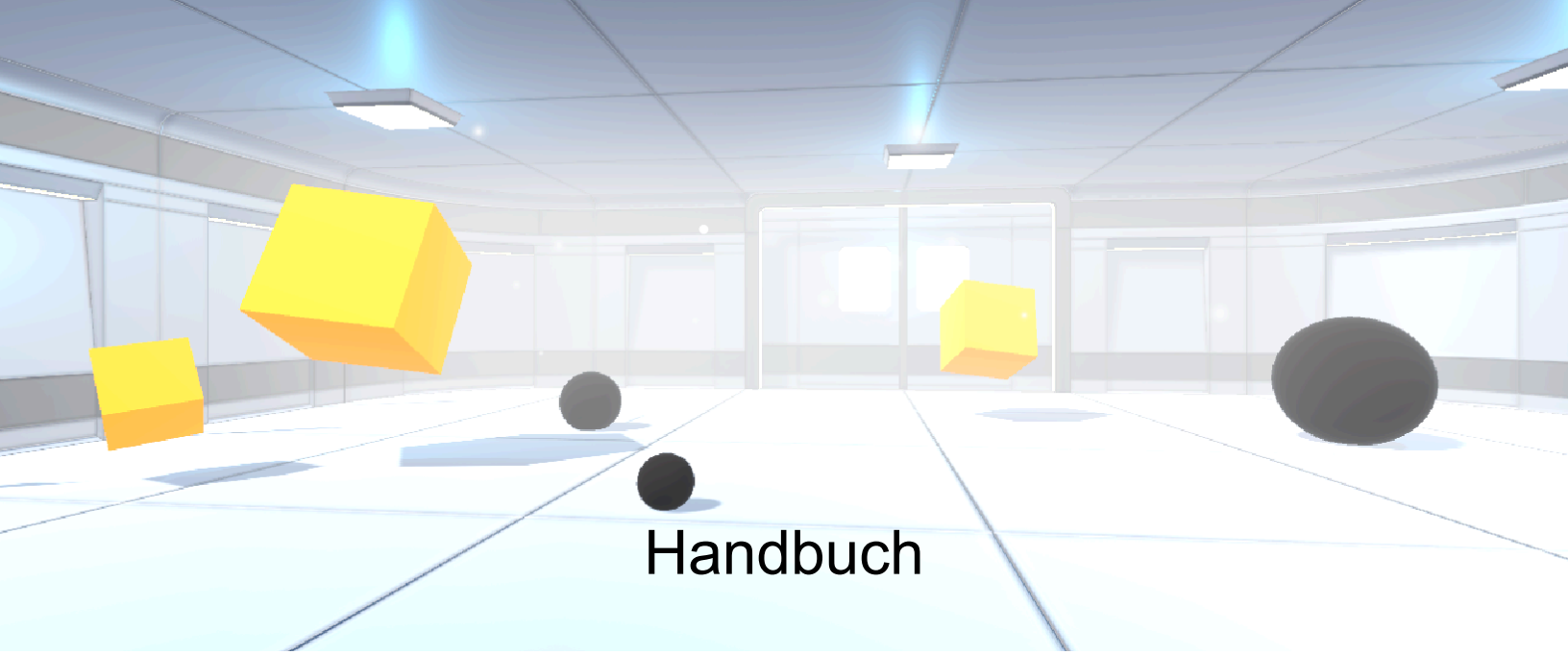

Handbuch

Implementierung visueller Illusionen im Game Design Anwendungsbeispiele

Ein Leitfaden zur Integration von räumlichen, Größen-, perspektivischen und psychologischen Illusionen in interaktive Umgebungen

Vorbereitet von:
Shirin Erol

Internationale Medieninformatik B.Sc. (ISMI)

Hochschule Bremen

Fakultät 4: Elektrotechnik und Informatik

Im Rahmen der Bachelorarbeit zu: Visuelle Illusionen in Videospielen: Analyse, Prototypentwicklung und Handbuch zur Steigerung des Spielerlebnisses

GitHub Link: https://github.com/Shaygi/Visual-Illusions.git

Bremen

Stand: 25. März 2025

Legende

Kategorisierung der Illusionen

Interaktive Illusionen
Diese Illusionen integrieren direkte Interaktionen des Spielers. Sie erfordern oft komplexe Skripte und eine enge Verzahnung von Gameplay und visuellen Effekten.

Klassische Illusionen
Diese Illusionen sind als optische Täuschungen gut bekannt. Sie demonstrieren, wie Wahrnehmung durch die Anordnung von Objekten und deren Proportionen manipuliert werden kann.

Basis Illusionen
Hierbei handelt es sich um grundlegende visuelle Effekte, die in verschiedenen Kontexten eingesetzt werden. Diese Illusionen sind oft weniger komplex in der Implementierung, aber essentiell für das Gesamtbild.

Komplexitäts Skala

- **1 – Einfach:**
  Minimaler Implementierungsaufwand mit Standard-Skripten und Assets.

- **2 – Moderat:**
  Erfordert einige Anpassungen und Feinabstimmungen, jedoch ohne tiefgehende technische Modifikationen.

- **3 – Fortgeschritten:**
  Komplexere Mechaniken, die spezielle Skripte oder benutzerdefinierte Shader erfordern.

- **4 – Komplex:**
  Hoher Entwicklungsaufwand, umfangreiche Tests und Optimierung sind notwendig.

- **5 – Sehr komplex:**
  Sehr anspruchsvolle Implementierungen, die fortgeschrittene Programmierung, Performance-Optimierung und möglicherweise externe Tools erfordern.

Inhaltsverzeichnis

Allgemeine Informationen....3
Genutzte Assets.... 3
Unity Features und Packages....3
1 Perspektivische Illusionen....4
1.1 Technische Implementierung....4
1.1.1 Perspektivische Größenveränderung/-skalierung (4).... 4
1.1.2 Ames-Raum (2)....9
1.1.3 Spiegelreflexion (4).... 15
1.2 Allgemeine Tipps & Warnungen.... 18
1.3 Verwendungszweck.... 19
2 Räumliche Illusionen....20
2.1 Technische Implementierung.... 20
2.1.1 Teleportation.... 20
2.1.1.1 Direkte Teleportation (1)....20
2.1.1.2 Nichteuklidische Portale (4).... 23
2.1.2 Dynamische und interaktive Raumveränderung....28
2.1.2.1 Dynamischer Raum (3).... 28
2.1.2.2 Trigger-Mechanik (2)....31
2.1.2.3 Objekte Toggeln (1)....35
2.1.2.4 Objekte Bewegen (2).... 38
2.1.3 Räumliche Tiefe (1)....43
2.4 Allgemeine Tipps & Warnungen.... 46
2.5 Verwendungszweck.... 47
3 Größen Illusionen.... 48
3.1 Technische Implementierung.... 48
3.1.1 Proportion & Positionierung.... 48
3.1.1.1 Ebbinghaus-Illusion (1).... 48
3.1.1.2 Bewusste Positionierung (2).... 50
3.1.2 Dynamische Größenveränderung (2).... 52
3.1.3 Licht und Farbe (1)....55
3.2 Allgemeine Tipps & Warnungen.... 57
3.3 Verwendungszweck.... 58
4 Psychologische Illusionen....59
4.1 Technische Implementierung....59
4.1.1 Tunnelblick (2).... 59
4.1.2 Wärmeempfinden (3).... 61

4.1.2.1 Profil....61
4.1.3.2 Post Processing Effekt....63
4.1.3 Kälteempfinden (2)....67
4.1.3.1 Profil....67
4.1.4 Schmerzempfinden (2)....71
4.2 Allgemeine Tipps & Warnungen....76
4.3 Verwendungszweck....78
Quellenverzeichnis....80
Abbildungsverzeichnis....83

Allgemeine Informationen

Dieses Projekt wurde in Unity 2022.3.10f1 entwickelt und basiert auf einem 3D-Projekt, das die Built-In Render Pipeline nutzt. Ziel des Projekts ist es, verschiedene visuelle Illusionstechniken zu implementieren und zu demonstrieren. Dabei wird ausschließlich auf kostenfreie Assets zurückgegriffen, um eine zugängliche und kosteneffiziente Grundlage zu schaffen.

Genutzte Assets

- *Free Modular Kit* – für die flexible Raumgestaltung
- *AllSky Free - 10 Sky / Skybox Set* – zur Erstellung realistischer Skyboxen
- *Fantasy Forest Environment* – als zusätzliche Dimension für nichteuklidische Portale
- *Mini First Person Controller* – zur Implementierung des First-Person-Controllers
- *VFX URP - Fire Package* – für feuereffekte

Unity Features und Packages

Das Projekt nutzt eine Vielzahl an Unity-Features und -Packages, die in die Entwicklung integriert sind. Zu den wichtigsten gehören:

- **Unity Features:** Engineering
- **Unity Packages:**
    - Code Coverage (locked – Teil des Feature-Sets)
    - Custom NUnit (als Abhängigkeit installiert)
    - Editor Coroutines (locked – Teil des Feature-Sets)
    - JetBrains Rider Editor (locked – Teil des Feature-Sets)
    - PostProcessing
    - Profile Analyzer (locked – Teil des Feature-Sets)
    - Settings Manager (als Abhängigkeit installiert)
    - Sysroot Base (als Abhängigkeit installiert)
    - Sysroot Linux x64 (als Abhängigkeit installiert)
    - Test Framework (locked – Teil des Feature-Sets)
    - TextMeshPro
    - Timeline
    - Toolchain Win Linux x64
    - Unity UI
    - Version Control
    - Visual Scripting
    - Visual Studio Code Editor (locked – Teil des Feature-Sets)
    - Visual Studio Editor (locked – Teil des Feature-Sets)

1 Perspektivische Illusionen

1.1 Technische Implementierung

1.1.1 Perspektivische Größenveränderung/-skalierung (4)

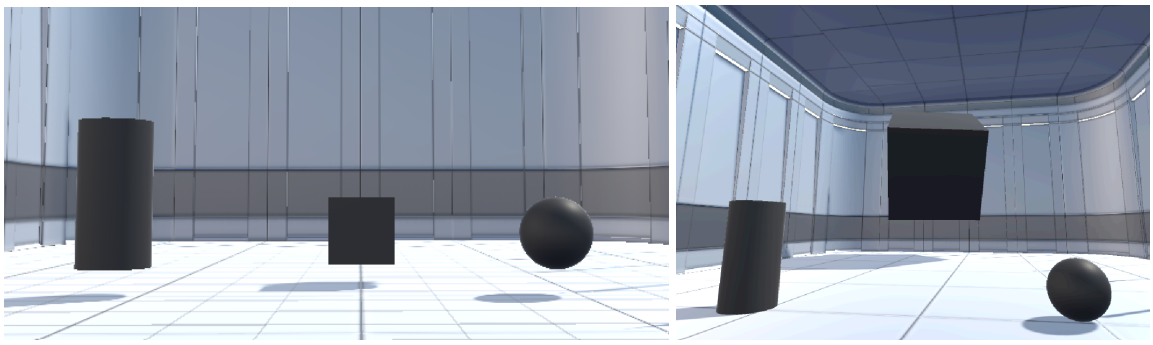

Abbildung 1: Perspektivische Größenveränderung

Umsetzung:

Spieler:

- Füge das Skript „PerspektiveResizingOverlap2“ der Spieler-Kamera hinzu.
  (Pfad: Assets → Scripts → PerspectiveIllusion → PerspektiveResizingOverlap2)
- Das Skript steuert die perspektivische Skalierung, sodass Objekte in Abhängigkeit von ihrer Entfernung zur Kamera dynamisch skaliert werden.

Objekte:

- Erstelle einen neuen Layer namens „Targetable“.
- Weise alle Objekte, die der Größenanpassung unterliegen sollen, diesem Layer zu.

Skript:

PerspektiveResizingOverlap2.cs

Pfad: Assets → Scripts → PerspectiveIllusion → PerspektiveResizingOverlap2.cs

```
    void HandleInput()
    {
        if (Input.GetMouseButtonDown(0))
```

```
        {
            if (target == null)
            {
                RaycastHit hit;
                if (Physics.Raycast(transform.position, transform.forward, out hit,
                Mathf.Infinity, targetMask))
                {
                    target = hit.transform;
                    target.GetComponent<Rigidbody>().isKinematic = true;
                    originalDistance = Vector3.Distance(transform.position,
                    target.position);
                    originalScale = target.localScale.x;
                    targetScale = target.localScale;
                }
            }
            else
            {
                target.GetComponent<Rigidbody>().isKinematic = false;
                target = null;
            }
        }
    }
```

Überprüfung:

- Prüft ob die linke Maustaste betätigt wurde

Wenn kein Zielobjekt (target) gesetzt ist:

- Führt einen Raycast in Blickrichtung (transform.forward) von der aktuellen Position (transform.position) aus, der nach einem Treffer in dem angegebenen Layer (targetMask) sucht.
- Wird ein Objekt getroffen, wird:
    - Das getroffene Objekt als Ziel (target) festgelegt.
    - Die Physik des Zielobjekts (Rigidbody) deaktiviert, indem isKinematic auf true gesetzt wird, um unbeabsichtigte physikalische Reaktionen zu vermeiden.
    - Die ursprüngliche Entfernung zwischen der Kamera und dem Ziel (originalDistance) berechnet und gespeichert.
    - Die ursprüngliche Skalierung des Zielobjekts (originalScale) anhand der x-Komponente gespeichert.
    - Die aktuelle Skalierung des Objekts (targetScale) initialisiert.

Falls bereits ein Ziel gesetzt ist:

- Wird die Physik des Zielobjekts wieder aktiviert (isKinematic auf false gesetzt).
- Das Zielobjekt (target) wird zurückgesetzt (auf null).

```
    void ResizeTarget()
{
    if (target == null) return;

    BoxCollider boxCollider = target.GetComponent<BoxCollider>();
    if (boxCollider == null)
    {
        Debug.LogWarning("Kein BoxCollider am Zielobjekt gefunden!");
        return;
    }

    Vector3 halfExtents = Vector3.Scale(boxCollider.size * 0.f,
    target.lossyScale);
    RaycastHit hit;
    if (Physics.Raycast(transform.position, transform.forward, out hit,
    Mathf.Infinity, ignoreTargetMask))
    {
        Vector3 desiredPosition = hit.point - transform.forward * offsetFactor
        * targetScale.x;
        target.position = Vector3.Lerp(target.position, desiredPosition,
        collisionLerpSpeed);
        for (int i = 0; i < collisionResolutionIterations; i++)
        {
            Collider[] overlaps = Physics.OverlapBox(target.position,
            halfExtents, target.rotation, ignoreTargetMask);
            foreach (Collider col in overlaps)
            {
                if (col.transform == target) continue;
                Vector3 direction;
                float distance;
                if (Physics.ComputePenetration(boxCollider, target.position,
                target.rotation,
                col, col.transform.position,
                col.transform.rotation,
                out direction, out distance))
                {
                    target.position += direction * distance;
                }
            }
        }
        float currentDistance = Vector3.Distance(transform.position,
        target.position);
        float s = currentDistance / originalDistance;
        targetScale = new Vector3(s, s, s);
        target.localScale = targetScale * originalScale;
    }
}
```

```
}
```

Zielüberprüfung:

- Wenn kein Zielobjekt (target) gesetzt ist, wird die Methode sofort beendet.

Collider-Prüfung:

- Es wird geprüft, ob das Zielobjekt einen BoxCollider besitzt.
- Falls kein BoxCollider gefunden wird, wird eine Warnung ausgegeben und die Methode beendet.

Berechnung der Halbextents:

- Die halben Abmessungen (halfExtents) des BoxColliders werden unter Berücksichtigung der Skalierung des Zielobjekts berechnet.

Raycast:

- Ein Raycast wird von der Kameraposition in Blickrichtung ausgeführt (unter Verwendung von ignoreTargetMask).
- Falls der Raycast ein Ziel trifft, wird der Punkt als Referenz für die gewünschte Position genutzt.

Positionierung:

- Die gewünschte Position wird als der getroffene Punkt minus einem Offset (basierend auf der Blickrichtung, offsetFactor und der aktuellen Skalierung) berechnet.
- Die Position des Zielobjekts wird mittels Lerp-Funktion sanft an diese gewünschte Position angepasst.

Kollisionsauflösung:

- Für eine definierte Anzahl von Iterationen werden alle Collider im Umfeld des Zielobjekts (überlapbox) überprüft.
- Falls eine Kollision festgestellt wird, wird der Penetrationsvektor berechnet und das Zielobjekt entsprechend verschoben, um Überschneidungen zu vermeiden.

Skalierungsberechnung:

- Die aktuelle Entfernung zwischen der Kamera und dem Zielobjekt wird ermittelt.
- Ein Skalierungsfaktor sss wird als Verhältnis der aktuellen Entfernung zur ursprünglichen Entfernung berechnet.

Anpassung der Skalierung:

- Das Zielobjekt wird in alle Richtungen mit dem Faktor sss skaliert, multipliziert mit der ursprünglichen Skalierung, um die perspektivische Größenveränderung zu realisieren.

Wichtige Hinweise:

- **Komponenten-Integration:**
  - Das Skript „PerspektiveResizingOverlap2“ sollte immer als Komponente an der Spieler-Kamera angebracht werden, um die Skalierungslogik korrekt zu implementieren. Dies gewährleistet, dass die Größenveränderung in Echtzeit auf den Spieler reagiert und der Illusionseffekt nahtlos dargestellt wird.
- **Regelmäßige Tests und Anpassungen:**
  - Es ist wichtig, den Code regelmäßig während der Entwicklung zu testen und Parameter wie den Offset-Faktor, die Anzahl der Kollisionsauflösungs-Iterationen und die Geschwindigkeit der Lerp-Funktion anzupassen. Diese Anpassungen stellen sicher, dass die Größenveränderung flüssig und realistisch bleibt, insbesondere wenn Objekte im Spielbereich groß oder klein skaliert werden.
- **Optimierung der Kollisionen:**
  - Um Überschneidungen von Objekten mit der Umgebung zu vermeiden, ist eine präzise Kollisionsauflösung erforderlich. Das Skript sieht drei Lösungsansätze vor, um Kollisionen zu minimieren, indem die Position des Objekts dynamisch angepasst wird. Es wird empfohlen, den Prozess zur Kollisionsauflösung detailliert zu testen und sicherzustellen, dass keine visuellen Artefakte entstehen, wenn Objekte die Geometrie durchdringen.
- **Objektlayer:**
  - Alle Objekte, die in ihrer Größe verändert werden sollen, müssen einem spezifischen Layer (z. B. „Targetable“) zugewiesen werden. Dies ermöglicht es dem Raycast, die betroffenen Objekte effizient zu erkennen und sicherzustellen, dass nur die beabsichtigten Objekte skaliert werden.
- **Skalierungsberechnung:**
  - Der Skalierungsfaktor (sss) wird dynamisch basierend auf der Entfernung zwischen dem Spieler und dem Objekt berechnet. Um eine übermäßig verzerrte Größenveränderung zu verhindern, sollte der Skalierungsfaktor innerhalb sinnvoller Grenzen (z. B. minScale und maxScale) geclamped werden, um ein realistischeres Verhalten zu gewährleisten.
- **Fehlende Collider:**
  - Die Methode prüft, ob das Zielobjekt einen BoxCollider besitzt, da die Kollisionserkennung für die Positionierung und Skalierung essenziell ist. Falls kein Collider vorhanden ist, wird eine Warnung im Debug-Log ausgegeben. Achte darauf, dass alle relevanten Objekte einen BoxCollider haben, um unnötige Fehler oder Probleme bei der Kollisionsberechnung zu vermeiden.
- **Performanceoptimierung:**
  - Das Skript zur Kollisionsprüfung und Größenveränderung kann bei einer großen Anzahl von betroffenen Objekten performanceintensiv sein. Eine Optimierung

besteht darin, die Anzahl der Iterationen zur Kollisionsauflösung und die Abtastrate der Raycasts in Abhängigkeit von der Szene und den Anforderungen des Spiels anzupassen.

- **Benutzerinteraktion:**
  - Die Steuerung der Größenveränderung erfolgt über die linke Maustaste. Es sollte darauf geachtet werden, dass das Benutzererlebnis durch eine intuitive Interaktion gefördert wird. Eine visuelle Rückmeldung beim Greifen und Loslassen der Objekte kann helfen, das Spielerlebnis zu verbessern und Missverständnisse bei der Nutzung zu vermeiden.

Genutzte Tutorials und Assets:

- **Tutorial**:
  - RumpledCode. (02.12.2020). Superliminal core scaling mechanic - Unity Tutorial [Video]. YouTube. Verfügbar unter: https://youtu.be/BNYy6O_Uy2Y?si=bxZtUIwBPTUSQAx9

Empfehlungen:

- Mujj. (01.05.2021). Remaking Superliminal's Mindbending Illusions… [Video]. YouTube . Verfügbar unter: https://youtu.be/QogeuKwlPNs?si=rZbs1A97GCz9WILh

---

1.1.2 Ames-Raum (2)

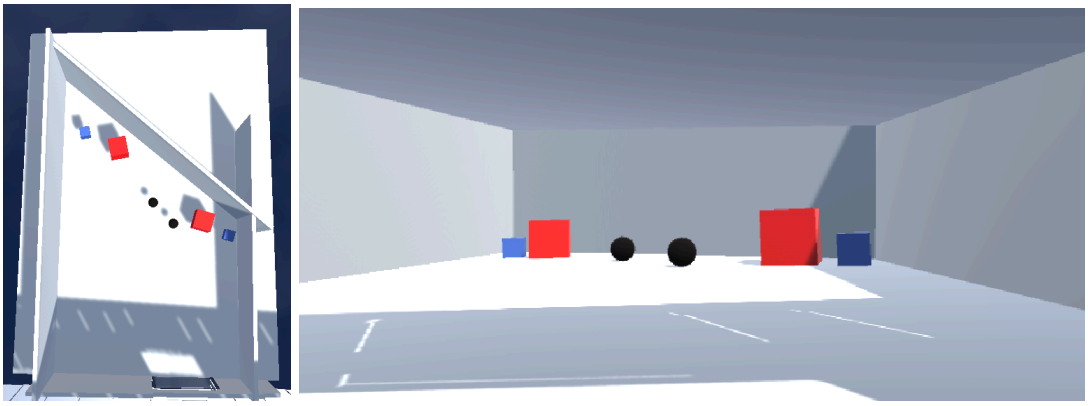

Abbildung 2: Ames-Raum

Umsetzung:

Raum:

- **Grundstruktur erstellen:**
  - **Boden und Decke:**
    - Erstelle Plane-Objekte für den Boden und die Decke.
    - Passe die Höhe und Länge des Raumes entsprechend an.

    - **Wände:**
        - Verwende Würfel, um die Wände zu modellieren.
        - Platziere die rechte und linke Wand parallel zueinander, um einen gleichmäßigen Raum zu erzeugen.
- **Raumverzerrung erzeugen:**
    - **Rotation der hinteren Bereiche:**
        - Rotiere den Boden und die Decke im hinteren Bereich (in der verlängerten Ecke des Raumes) so, dass die Rückwand optisch die gleiche Höhe aufweist wie die Wand an der kürzeren Seite.
        - Diese gezielte Drehung bewirkt, dass der Raum verzerrt wirkt und Objekte in den hinteren Ecken unterschiedlich wahrgenommen werden.

Raum zu finden unter: Scenes → Perspektivische_Täuschung → Ames Room

Objekte:

- **Objekte platzieren:**
    - Platziere zwei identische Objekte – eines in der linken hinteren Ecke und eines in der rechten hinteren Ecke.
    - Durch die verzerrte Raumstruktur sollten beide Objekte, obwohl gleich groß, unterschiedlich wahrgenommen werden.
- **Interaktivität hinzufügen:**
    - Hänge das ObjektPickup-Skript an die Objekte, sodass der Nutzer sie selbst arrangieren und die Illusion experimentell erleben kann
        - Pfad: Assets → Scripts → PlayerInteraction → ObjectPickup.cs

---

Skript:

Pfad: Assets → Scripts → PlayerInteraction → ObjectPickup.cs

```
    void Update()
    {
        if (Input.GetMouseButtonDown(0))
        {
            if (heldObj == null)
            {
                Ray ray = playerCamera.ViewportPointToRay(new Vector3(0.5f,
                0.5f, 0));
                if (Physics.Raycast(ray, out RaycastHit hit, pickupRange))
                {
                    if (((1 << hit.collider.gameObject.layer) &
                    moveableLayer.value) != 0)
```

```
                {
                    heldObj = hit.collider.gameObject;
                    heldRb = heldObj.GetComponent<Rigidbody>();

                    if (heldRb != null)
                    {
                        heldJoint =
                        heldObj.AddComponent<ConfigurableJoint>();
                        heldJoint.autoConfigureConnectedAnchor = false;

                        Rigidbody holdRb =
                        holdParent.GetComponent<Rigidbody>();
                        heldJoint.connectedBody = holdRb;

                        heldJoint.anchor = Vector3.zero;
                        heldJoint.connectedAnchor = Vector3.zero;
                        heldJoint.targetPosition = Vector3.zero;

                        heldJoint.angularXMotion =
                        ConfigurableJointMotion.Locked;
                        heldJoint.angularYMotion =
                        ConfigurableJointMotion.Locked;
                        heldJoint.angularZMotion =
                        ConfigurableJointMotion.Locked;

                        heldJoint.xMotion =
                       ConfigurableJointMotion.Free;
                       heldJoint.yMotion =
                       ConfigurableJointMotion.Free;
                       heldJoint.zMotion =
                       ConfigurableJointMotion.Free;


                       JointDrive drive = new JointDrive();
                       drive.positionSpring = positionSpring;
                       drive.positionDamper = positionDamper;
                       drive.maximumForce = maximumForce;
                       heldJoint.xDrive = drive;
                       heldJoint.yDrive = drive;
                       heldJoint.zDrive = drive;

                       heldRb.freezeRotation = true;
                    }
```

```
                }
            }
        }
        else
        {
            if (heldJoint != null)
            {
                Destroy(heldJoint);
            }
            if (heldRb != null)
            {
                heldRb.freezeRotation = false;
            }
            heldObj = null;
            heldRb = null;
        }
    }
}
```

Interaktionslogik:

- Überprüft, ob der linke Mausknopf gedrückt wurde.

Objekt aufnehmen:

- **Falls aktuell kein Objekt gehalten wird (*heldObj* ist null):**
    - Erzeugt einen Ray von der Mitte des Bildschirms (Viewport-Center).
    - Führt einen Raycast aus, um zu prüfen, ob sich innerhalb von *pickupRange* ein Objekt befindet.
    - Überprüft, ob das getroffene Objekt zum *moveableLayer* gehört.
    - Weist *heldObj* auf das getroffene Objekt zu und speichert dessen Rigidbody in *heldRb*.
- **Falls ein Rigidbody vorhanden ist:**
    - **Fügt dem Objekt dynamisch einen ConfigurableJoint hinzu und konfiguriert diesen:**
        - Deaktiviert die automatische Konfiguration des verbundenen Ankers.
        - Verbindet den Joint mit dem Rigidbody des *holdParent*.
            - Setzt *anchor*, *connectedAnchor* und *targetPosition* auf Null.
        - Sperrt alle Winkelbewegungen (Rotationen) des Joints.
        - Erlaubt freie Bewegung entlang aller Achsen (Translation) und setzt dabei die JointDrive-Parameter (Feder, Dämpfung, maximale Kraft).
        - Friert die Rotation des aufgenommenen Objekts ein, um unkontrollierte Drehungen zu vermeiden.

Objekt ablegen:

- **Falls bereits ein Objekt gehalten wird:**
  - Entfernt den ConfigurableJoint vom gehaltenen Objekt, falls vorhanden.
  - Gibt die Rotationssperre des Rigidbody wieder frei.
  - Setzt *heldObj* und *heldRb* auf null, um anzuzeigen, dass kein Objekt mehr gehalten wird.

Wichtige Hinweise:

- **Konsistenz der Raumgeometrie:**
  - Es ist entscheidend, dass der Boden und die Decke trotz der geometrischen Verzerrungen aus der Perspektive des Betrachters parallel erscheinen. Achte darauf, dass die Drehung von Boden und Decke so konfiguriert wird, dass die Illusion der perspektivischen Verzerrung an den vorgesehenen Betrachtungspunkten optimal funktioniert. Eine fehlerhafte Rotation kann die Illusion beeinträchtigen.
- **Kollisionsvermeidung bei runden Objekten:**
  - Wenn runde Objekte oder solche mit dynamischer Physik (Rigidbody-Komponenten) in den Ames-Raum eingefügt werden, kann es vorkommen, dass diese Objekte durch die geneigten Flächen oder die geometrischen Verzerrungen in nicht beabsichtigte Bereiche rollen oder sich unvorhersehbar verhalten. Verwende zusätzlich Box Colliders, um diese Objekte in Position zu halten und ungewollte Bewegungen zu verhindern.
- **Interaktionslogik und Objektplatzierung:**
  - Stelle sicher, dass die Objekte, die zur Demonstration der Größenillusionen im Ames-Raum verwendet werden, korrekt ausgerichtet sind. Der Spieler sollte in der Lage sein, die Objekte mit dem ObjectPickup-Skript aufzunehmen und zu platzieren, ohne dass die Illusion durch unerwartete physikalische Reaktionen zerstört wird. Achte darauf, dass die Objekte nach dem Loslassen stabil und ohne Rotationsfehler positioniert bleiben.
- **Physikalische Eigenschaften der Objekte:**
  - Damit die Interaktivität mit den Objekten reibungslos funktioniert, sollte sichergestellt werden, dass jedes Objekt einen funktionalen Rigidbody besitzt. Dies ermöglicht das Aufnehmen und Platzieren der Objekte durch den Spieler. Achte darauf, dass der Configurable Joint richtig eingestellt ist, damit das Objekt in den richtigen Achsen frei beweglich bleibt, während die Rotationen gesperrt werden.
- **Performance-Optimierung bei der Objekterkennung:**
  - Verwende Layer Masken (z. B. „moveableLayer“) effizient, um sicherzustellen, dass nur relevante Objekte von der Interaktionslogik erfasst werden. Dies optimiert die Performance und verhindert, dass unerwünschte Objekte in die Physikberechnungen einbezogen werden.

- **Spielerzentrierte Erfahrung:**
  - Der Ames-Raum sollte so gestaltet sein, dass die Interaktion mit den Objekten und der Umgebung für den Spieler intuitiv und fließend ist. Das Hinzufügen von klaren Anweisungen oder visuellen Hilfen zur Interaktion mit den Objekten kann das Spielerlebnis verbessern und die Funktionsweise der Illusion verdeutlichen.
- **Sicherstellung der Funktionsfähigkeit in verschiedenen Kamera- und Spieleinstellungen:**
  - Stelle sicher, dass der Ames-Raum und die zugehörigen Interaktionen in unterschiedlichen Auflösungen und bei verschiedenen Sichtfeldern (FOV) der Kamera korrekt funktionieren. Unterschiedliche Kameraeinstellungen können die Wahrnehmung der Illusion beeinflussen, daher sollte die Illusion in verschiedenen Szenarien getestet werden.

Genutzte Tutorials und Assets:

- **Tutorial**:
  - SpeedTutor. (26.03.2022). PICKUP & DROP Physics Objects in Unity (Like PORTAL) [Video]. YouTube. Verfügbar unter: https://youtu.be/6bFCQqabfzo?si=329kJEtn8yTkZZZw
  - experimentis. Irre Täuschung: Der Ames-Raum [Webseite]. experimentis. Verfügbar unter: https://www.experimentis.de/experimente-versuche/optik-akustik/ames-room-ames-raum-basteln-adelbert-ames/

Empfehlungen:

- **Raumaufbau:**
  - d'Art of Science. (23.12.2016). AMES ROOM Illusion | REVEALED | dArtofScience [Video]. YouTube. Verfügbar unter: https://youtu.be/lrVWnDPybyc?si=usQQL2AuTc0s8ezo

---

1.1.3 Spiegelreflexion (4)

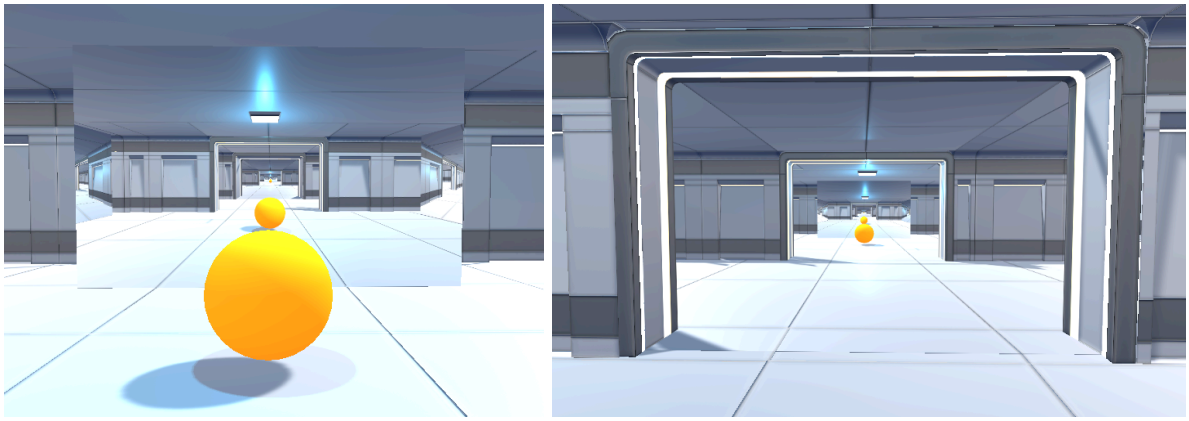

Abbildung 3: Spiegelreflexion

Umsetzung:

Spiegel:

- **Spiegel erstellen:**
  - Erstelle ein leeres GameObject als Basis für den Spiegel.
  - Füge diesem GameObject ein Plane-Objekt hinzu, das als sichtbare Spiegeloberfläche dient.
- **Rendertexture und Kamera:**
  - Erstelle eine Rendertexture mit denselben Seitenverhältnissen wie das Plane.
  - Erstelle im selben GameObject eine Kamera.
    - Rotiere die Kamera so, dass sie in Richtung der Vorderseite des Planes schaut.
    - Weisen der Kamera die Rendertexture als „Target Texture“ zu.
- **Material erstellen:**
  - Erstelle ein neues Material und übergebe diesem die zuvor erstellte Rendertexture.
  - Weise das Material dem Plane zu, sodass es den Spiegel-Effekt wiedergibt.
- **Feinjustierung der Kamera:**
  - Positioniere die Kamera hinter dem Plane, bis ihre Sichtlinien die Kanten des Planes berühren.
  - Passe die Clipping-Planes der Kamera so an, dass sie genau auf der Position des Planes liegen, um zu verhindern, dass die Kamera Inhalte hinter dem Plane „filmt“.
- **Spiegelung realisieren:**
  - Skaliere das Plane auf der X-Achse negativ (z. B. setze den Skalierungswert auf -1), um den gespiegelten Effekt zu erzielen.

- **Interaktive Rotation der Kamera:**
    - Erstelle ein weiteres leeres GameObject als Parent, in das du die Spiegel-Kamera verschiebst.
    - Erstelle ein Skript (z. B. „MirrorReflection“), das die Rotation der Kamera basierend auf der Position des Spielers anpasst.
        - Pfad: Assets → Scripts → PerspectiveIllusion → MirrorReflection
        - Das Skript berechnet anhand der Position der Spieler-Kamera und der Spiegelposition einen Winkel und setzt die lokale Y-Rotation der Spiegel-Kamera entsprechend.

Skript:

MirrorReflection.cs

Pfad: Assets → Scripts → PerspectiveIllusion → MirrorReflection.cs

```
    void Update()
    {
        Vector3 PosY = new Vector3(transform.position.x,
        PlayerCamera.transform.position.y, transform.position.z);

        Vector3 side1 = PlayerCamera.transform.position - PosY;

        Vector3 side2 = transform.forward;

        float angle = Vector3.SignedAngle(side1, side2, Vector3.up);

        MirrorCamera.localEulerAngles = new Vector3(0, angle, 0);
    }
}
```

Berechnung eines Referenzpunktes:

- Erzeugt einen Vektor (PosY) an der x- und z-Position des Spiegels, aber mit der y-Position der Spieler-Kamera.
    - Stellt sicher, dass nur die Höhe der Spieler-Kamera berücksichtigt wird.

Bestimmung des Relativvektors:

- Berechnet side1 als Vektor von PosY zur Position der Spieler-Kamera.
    - Dieser Vektor repräsentiert, wie weit und in welche Richtung sich die Spieler-Kamera relativ zum Spiegel befindet.

Definition der Spiegel-Normal:

- Nutzt transform.forward des Spiegels als side2.
    - Diese Richtung dient als Referenz, um den Winkel zwischen der Position der Spieler-Kamera und der Front des Spiegels zu bestimmen.

Winkelberechnung:

- Berechnet den signierten Winkel (angle) zwischen side1 und side2 um die y-Achse (Vector3.up).
    - Der Winkel gibt an, wie stark die Spieler-Kamera von der Spiegel-Normal abweicht.

Anpassung der Spiegel-Kamera-Rotation:

- Setzt die lokale Y-Rotation der Spiegel-Kamera (MirrorCamera.localEulerAngles) auf den berechneten Winkel, während X- und Z-Rotation auf 0 gesetzt werden.
    - Spiegel-Kamera wird korrekt ausgerichtet, um den Spiegel-Effekt dynamisch zu erzeugen.

Wichtige Hinweise:

- **Rekursion vermeiden:**
    - Bei der Arbeit mit Spiegelungen kann es leicht zu rekursiven Spiegelungen kommen, bei denen der Spiegel sich selbst reflektiert und eine Endlosschleife erzeugt. Um dies zu verhindern, sollte die maximale Anzahl der Spiegelungen begrenzt werden. Dies kann durch gezielte Begrenzung der Renderdepth in den Kameraeinstellungen oder durch das Abschalten der Spiegelung in Spiegeln innerhalb eines anderen Spiegels realisiert werden.
- **Performance-Optimierung:**
    - Jeder zusätzliche Spiegel fügt der Szene weitere Render-Operationen hinzu, die eine signifikante Belastung für die Rechenleistung darstellen können. Besonders bei der Verwendung von RenderTextures für die Spiegelungen sollten die Spiegel effizient verwaltet werden. Begrenze die Anzahl der aktiven Spiegel und RenderTextures in deiner Szene, um die Performance zu sichern. Zusätzlich kann die Qualität der RenderTexture an die Leistung angepasst werden, indem die Auflösung verringert wird.
- **Visuelle Artefakte minimieren:**
    - Bei mehrfachen Spiegelungen kann es zu visuellen Artefakten wie Unschärfen, Flimmern oder anderen Rendering-Fehlern kommen. Achte darauf, dass du geeignete Clipping-Planes einsetzt, um zu verhindern, dass die Kamera hinter die Spiegeloberfläche „sieht“. Außerdem sollten RenderTextures sorgfältig aktualisiert werden, um flackernde Bilder oder Glitches zu vermeiden. Ein

weiterer Ansatz zur Reduzierung von Artefakten kann die Anpassung des Anti-Aliasing sein.

- **Platzierung der Spiegel-Kamera und RenderTexture:**
    - Die korrekte Platzierung der Kamera und der RenderTexture ist entscheidend für die Illusion eines realistischen Spiegelbilds. Die Kamera sollte genau so positioniert werden, dass ihre Blickrichtung die exakte Perspektive eines echten Spiegels im Raum simuliert. Achte darauf, dass die Kamera die Spiegelkante nicht überschreitet, und passe die Clipping-Planes so an, dass die Sichtlinie nur den Bereich erfasst, den der Spiegel tatsächlich abbildet.
- **Auflösung und Detailgrad der Spiegelungen:**
    - Die Auflösung der RenderTexture sollte den Anforderungen der Szene angepasst werden. In weniger wichtigen Bereichen kann eine niedrigere Auflösung verwendet werden, um die Performance zu verbessern. In Szenen, in denen die Spiegelung von hoher visueller Qualität sein muss, sollte eine entsprechend hochauflösende RenderTexture verwendet werden, um Details klar wiederzugeben.

Genutzte Tutorials und Assets:

- **Tutorial**:
    - UnityDude. (02.08.2024). Crafting mirrors in Unity : Reflections Tutorial [Video]. YouTube. Verfügbar unter: https://youtu.be/_jBWm0KM2ec?si=R_q3Q2Icjz5jONbY

Empfehlungen:

- Game Dev Guru. (25.12.2022). Unity - How to make a mirror in Unity [2023 UPDATED] [Video]. YouTube. Verfügbar unter: https://youtu.be/RzT4m1-ltvU?si=yADNmgperDWvDvwN
- Yi Zhen. (letzte Veröffentlichung: 5.10.2024). Fantasy Mirror Primer [Asset]. Unity Asset Store. Verfügbar unter: https://assetstore.unity.com/packages/vfx/shaders/fantasy-mirror-primer-189252

1.2 Allgemeine Tipps & Warnungen

Kamera-Perspektive ist entscheidend:

- Perspektivische Täuschungen basieren vollständig auf der Position und dem Blickwinkel der Kamera. Achte darauf, dass die Kamera korrekt positioniert ist und sich der Spieler in einem festen, kontrollierten Rahmen bewegt, wenn die Illusionen wirken sollen. Andernfalls kann es passieren, dass der Effekt nicht wie gewünscht wahrgenommen wird.

Effekte variieren je nach Abstand:

- Die Wirkung von perspektivischen Täuschungen ändert sich mit der Entfernung des Spielers zur Szene. Gerade bei Größenveränderungen ist der Abstand der Kamera entscheidend für die Wahrnehmung der Objektgröße. Stelle sicher, dass der Spieler

einen gewissen Bewegungsfreiraum hat, um die Illusionen aus verschiedenen Winkeln zu erleben.

Konflikte zwischen Illusionen und Interaktion vermeiden:

- Wenn der Spieler die Objekte, die Teil der Illusion sind, manipulieren kann, solltest du sicherstellen, dass die Interaktion die Täuschung nicht auflöst. Zum Beispiel könnte ein Objekt, das perspektivisch verzerrt wird, durch eine Interaktion seine Größe ändern und die Illusion zerstören. Verwende entsprechende Einschränkungen oder zusätzliche Mechaniken, um diese Probleme zu umgehen.

Performance beachten:

- Perspektivische Täuschungen, insbesondere solche mit komplexen Mechaniken wie dynamischer Größenveränderung oder fortlaufenden Kameraanpassungen, können rechenintensiv sein. Achte darauf, dass die Optimierung im Blick behalten wird. Verwende Culling oder Layer, um unnötige Berechnungen zu vermeiden.

Kollisionsabfragen anpassen:

- Die Illusion könnte durch physikalische Interaktionen oder ungewollte Kollisionen gestört werden. Wenn Objekte perspektivisch stark verzerrt werden, passe die Kollisionsabfragen so an, dass sie das Objekt in seiner „wahren“ Größe berücksichtigen und keine unvorhersehbaren physikalischen Effekte verursachen.

Begrenzte Anwendungsbereiche:

- Perspektivische Täuschungen sind sehr ortsgebunden. Sie funktionieren in der Regel nur aus einem bestimmten Blickwinkel oder in bestimmten Bereichen des Spiels. Sorge dafür, dass der Spieler in den vorgesehenen Bereichen bleibt, um die Illusion bestmöglich zu erfahren.

Vermeidung von visuellen Artefakten:

- In Szenen mit vielen geometrischen Verzerrungen oder Größenveränderungen können visuelle Artefakte wie Glitches oder flimmernde Texturen auftreten. Teste die Szene sorgfältig und passe die Rendering-Einstellungen sowie die Texturgrößen an, um diese Artefakte zu minimieren.

Testen aus verschiedenen Blickwinkeln:

- Da perspektivische Täuschungen aus einem bestimmten Blickwinkel am effektivsten sind, ist es wichtig, die Szenen aus mehreren Perspektiven zu testen. Sorge dafür, dass die Illusion nicht nur aus einem festen Punkt funktioniert, sondern dass der Spieler auch in Bewegung die Täuschung wahrnehmen kann.

1.3 Verwendungszweck

Erkundung und Rätsel:

- Perspektivische Täuschungen eignen sich hervorragend für Rätselspiele, bei denen der Spieler durch geschicktes Manipulieren der Perspektive oder der Objektgrößen weiterkommt. Beispiele sind Spiele wie *Superliminal* oder *Portal*, in denen der Spieler Objekte perspektivisch manipuliert, um neue Wege zu öffnen oder Rätsel zu lösen.

Immersion und Erlebnisse:

- Durch perspektivische Täuschungen kann der Spieler tiefer in die Spielwelt eintauchen. Indem die Größenverhältnisse oder Sichtweisen dynamisch angepasst werden, können Spieler ein Gefühl von Staunen oder Verwirrung erleben, was die Immersion im Spiel verstärkt

Optische Illusionen als visuelles Highlight:

- Perspektivische Täuschungen sind oft visuell beeindruckend und können als stilistisches Mittel in Spielen eingesetzt werden, um besondere Momente zu kreieren. Diese Illusionen eignen sich besonders in Szenen, die den Spieler überraschen oder beeindrucken sollen, beispielsweise indem sie unerwartete Größenunterschiede oder unmögliche Geometrien darstellen.

Versteckte Räume und Strukturen:

- Diese Illusionen sind ideal, um versteckte Räume oder Geheimnisse zu verbergen, die nur aus einer bestimmten Perspektive sichtbar sind. Der Spieler wird dazu angeregt, die Welt genau zu erkunden und seine Umgebung aufmerksam zu betrachten, um versteckte Inhalte zu entdecken.

Psychologische Effekte:

- Perspektivische Täuschungen können verwendet werden, um beim Spieler ein Gefühl der Unsicherheit oder Desorientierung zu erzeugen. Dies ist besonders in psychologischen Horrorspielen von Vorteil, wo die Verzerrung der Wahrnehmung den Spieler verunsichert und die Spannung erhöht.

Künstlerische Freiheit:

- Entwickler können durch perspektivische Illusionen kreative Freiheit genießen und Szenen erschaffen, die physikalisch unmöglich wären. Dadurch lassen sich ästhetische und künstlerische Aspekte in das Spieldesign integrieren, die dem Spieler ein einzigartiges visuelles Erlebnis bieten.

Unterstützung von Storytelling:

- Perspektivische Täuschungen können auch im Storytelling eine Rolle spielen, etwa um Metaphern oder symbolische Darstellungen zu unterstützen, bei denen die Realität aus verschiedenen Blickwinkeln anders erscheint.

2 Räumliche Illusionen

2.1 Technische Implementierung

2.1.1 Teleportation

2.1.1.1 Direkte Teleportation (1)

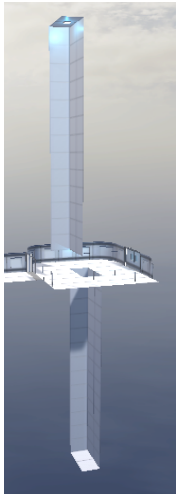

Abbildung 4: Teleportation im freien Fall

Umsetzung:

Tunnel:

- Lege zwei Tunnel aus einfachen Würfeln oder gegebenen Assets an

Trigger:

- zwei horizontale Planes (Eingang und Ausgang) erstellen und im Box Collider ‘is Trigger’ auswählen
    - material entfernen damit das plane nicht sichtbar ist
- Skript anfügen an den Eingang und das andere pläne für die transformation übergeben
    - Pfad: Assets → Scripts → SpatialIllusion → FallTeleporter.cs
- Platziere die Trigger in der mitte der beiden Tunnel

Skript:

FallTeleporter.cs

Pfad: Assets → Scripts → SpacialIllusion → FallTeleporter.cs

```
    private void OnTriggerEnter(Collider other)
    {
        if (other.CompareTag("Player"))
        {
            other.transform.position = target.position;
        }
```

```
    }
```

- Die Methode OnTriggerEnter(Collider other) wird aufgerufen, wenn ein Collider den Triggerbereich betritt.
- Es wird überprüft, ob das eintretende Objekt (other) das Tag "Player" besitzt.
- Ist dies der Fall, wird die Position des Spielers (other.transform.position) auf die Position des Zielobjekts (target.position) gesetzt.

Wichtige Hinweise:

- **Unauffällige Platzierung der Planes:**
  - Die beiden Planes (Eingang und Ausgang) sollten so positioniert werden, dass der Übergang für den Spieler möglichst fließend und unauffällig wirkt. Dies ist besonders wichtig, wenn das Ziel darin besteht, den Spieler zu verwirren oder ihn subtil in eine andere Umgebung zu befördern, ohne dass er die Teleportation direkt bemerkt.
- **Visuelle Konsistenz der Tunnel:**
  - Um die Illusion zu verstärken, sollten die Tunnel optisch identisch oder sehr ähnlich gestaltet werden. Dies minimiert visuelle Unterschiede, die den Spieler auf den Wechsel aufmerksam machen könnten. Achte darauf, dass auch die Beleuchtung und Texturen konsistent sind, um die Teleportation weniger spürbar zu machen.
- **Ästhetische Gestaltung mit Post-Processing-Effekten:**
  - Um die Teleportation visuell interessanter zu gestalten, können Post-Processing-Effekte wie Blenden, Vignettierung oder Blur eingesetzt werden. Diese Effekte lenken den Spieler von der tatsächlichen Positionsänderung ab und erhöhen gleichzeitig die ästhetische Qualität der Teleportation.
- **Abrupte Teleportation für dramatische Effekte:**
  - Alternativ kann die Teleportation bewusst abrupt und offensichtlich gestaltet werden, um den Spieler gezielt in neue Räume oder Umgebungen zu befördern. Dies eignet sich besonders für narrative oder dramatische Momente, in denen der Spieler überrascht oder desorientiert werden soll.
- **Sicherstellen der Trigger-Funktionalität:**
  - Teste den Triggermechanismus gründlich, um sicherzustellen, dass der Spieler zuverlässig teleportiert wird, sobald er den Eingangsbereich betritt. Achte darauf, dass der Box Collider korrekt eingestellt ist und keine unerwünschten Objekte den Teleportationsprozess auslösen.
- **Erweiterbarkeit der Teleportation:**
  - Die einfache Teleportation kann leicht erweitert werden, um zusätzliche Features wie Effekte oder Animationen einzubauen. Beispielsweise könnten während der

Teleportation Klang- oder Lichteffekte eingesetzt werden, um die Illusion intensiver zu gestalten.

- **Teleportation zur Irreführung des Spielers:**
  - Diese Methode kann besonders effektiv eingesetzt werden, um den Spieler in die Irre zu führen, indem er ohne Vorwarnung in einen anderen Bereich des Spiels versetzt wird. Dies lässt sich nutzen, um das Raumverständnis des Spielers zu manipulieren und den Eindruck unendlicher Korridore oder nicht-euklidischer Geometrien zu erzeugen.
- **Optimierung für Performance:**
  - Da die Teleportation keine großen Berechnungen erfordert, ist sie sehr ressourcenschonend. Dennoch sollte sichergestellt werden, dass überflüssige Teleportationsvorgänge (wie versehentliche Rücksprünge) vermieden werden, indem klare Trigger-Bedingungen definiert werden.

2.1.1.2 Nichteuklidische Portale (4)

Abbildung 5: Nichteuklidische Portale

Umsetzung:

Portale:

- Erstelle zwei leere GameObjects (z. B. PortalA und PortalB) und baue einen Rahmen aus einfachen Würfeln.
- Verwende eine Plane als inneren Bereich des Portals.

Kameras:

- Setze zwei Kameras ein (PortalCameraA und PortalCameraB) und hänge ein „PortalCameraController“-Skript an.
- Füge dem Skript die Eingangs- und Ausgangsportale sowie die Spieler-Kamera hinzu.

Materialien:

- Erstelle zwei Texturen und Materialien für beide Portale und weise den Materialien den Unlit/PortalShader zu.
- Alternativ kann ein GameManager mit einem PortalTexture-Skript genutzt werden, um die Texturen dynamisch zu erstellen.

Trigger-Collider & Teleportation:

- Füge den Portalen einen weiteren Plane als Trigger-Collider hinzu.
    - entferne den Mesh Renderer um es unsichtbar zu machen
- Füge das Teleporter Skript als Komponente hinzu
    - Pfad: Assets → Scripts → SpacialIllusion → Teleporter.cs

---

Skript:

Teleporter.cs

Pfad: Assets → Scripts → SpacialIllusion → Teleporter.cs

```
void Update()
{
        if (isOverlapping)
    {
        Vector3 portalToPlayer = player.position - transform.position;
        float dotProduct = Vector3.Dot(transform.up, portalToPlayer);
        if (dotProduct > 0f)
        {
            float rotationDifference =
            -Quaternion.Angle(transform.rotation, reciever.rotation);
            rotationDifference += 180;

            player.Rotate(Vector3.up, rotationDifference);

            Vector3 positionOffset = Quaternion.Euler(0f,
            rotationDifference, 0f) * portalToPlayer;
```

```
            player.position = reciever.position + positionOffset;
            isOverlapping = false;
        }
    }
}
```

Überprüfung:

- Prüft, ob der Spieler den Teleportbereich betritt (Variable isOverlapping ist true).

Berechnung des Vektors:

- Berechnet den Vektor portalToPlayer vom Portal (aktuelles Objekt) zur Spielerposition.

Skalarprodukt:

- Ermittelt das Skalarprodukt zwischen dem Aufwärtsvektor des Portals (transform.up) und portalToPlayer.
- Ist das Skalarprodukt positiv, wurde das Portal vollständig überquert.

Rotation anpassen:

- Berechnet den Winkelunterschied zwischen der Rotation des Portals und der Rotation des Empfängerportals (reciever).
- Der berechnete Winkel wird um 180 Grad angepasst, um die korrekte Ausrichtung sicherzustellen.
- Der Spieler wird um diesen Winkel (um die Y-Achse) rotiert.

Positionsanpassung:

- Ein positionsversetzter Vektor (positionOffset) wird basierend auf der Rotation berechnet.
- Die neue Position des Spielers wird als Summe aus der Position des Empfängers und dem Offset gesetzt.

Abschluss:

- Nach dem Teleportieren wird isOverlapping auf false gesetzt, um den Vorgang abzuschließen.

```
private void OnTriggerEnter(Collider other)
{
        if (other.tag == "Player")
    {
        isOverlapping = true;
```

```
    }
}

private void OnTriggerExit(Collider other)
{
    if (other.tag == "Player")
    {
        isOverlapping = false;
    }
}
```

OnTriggerEnter:

- Wird aufgerufen, wenn ein Collider in den Triggerbereich eintritt.
- Überprüft, ob das eintretende Objekt das Tag „Player“ besitzt.
- Falls ja, wird isOverlapping auf true gesetzt, um anzuzeigen, dass der Spieler den Teleportbereich betritt.

OnTriggerExit:

- Wird aufgerufen, wenn ein Collider den Triggerbereich verlässt.
- Überprüft, ob das verlassende Objekt das Tag „Player“ besitzt.
- Falls ja, wird isOverlapping auf false gesetzt, um anzuzeigen, dass der Spieler den Bereich verlassen hat.

Wichtige Hinweise:

- **Skalierung und Positionierung der Portale:**
    - Die Portale müssen präzise zueinander ausgerichtet und skaliert werden, damit der Spieler keine sichtbaren Fehler oder Unstimmigkeiten bei der Teleportation wahrnimmt. Achte darauf, dass die Portale während der Implementierung korrekt ausgerichtet sind und keine Kollisionen mit anderen Objekten verursachen.
- **Kamerawinkel und Sichtfeld:**
    - Die Kameras müssen sicherstellen, dass das Portal immer aus der richtigen Perspektive angezeigt wird. Überprüfe, dass das Sichtfeld der Kameras so eingestellt ist, dass der Spieler das Portal und den Übergang klar sehen kann. Eine falsche Kameraeinstellung kann zu visuellen Glitches führen, wenn der Spieler hindurchtritt.
- **Performance-Optimierung:**
    - Portale können die Performance erheblich beeinflussen, insbesondere wenn sie große Bereiche darstellen oder komplexe Shader verwendet werden. Achte darauf, dass die Portal-Kameras nur den Bereich rendern, den der Spieler durch

das Portal sieht, um unnötige Berechnungen und Rendering-Prozesse zu vermeiden. Verwende ggf. LOD (Level of Detail) für entfernte Objekte.

- **Shader-Optimierung und Anpassung:**
  - Die verwendeten Shader (wie der Unlit/PortalShader) sollten effizient und für die Teleportation optimiert sein. Überprüfe, dass keine unerwünschten Artefakte auftreten, wie z. B. flimmernde Texturen oder ungenaue Portal-Darstellungen bei schnellen Bewegungen.
- **Benutzerfeedback:**
  - Es kann hilfreich sein, visuelles oder akustisches Feedback zu integrieren, wenn der Spieler sich dem Portal nähert. Dies kann durch Übergangseffekte oder ein leichtes "Ziehen" des Bildes erfolgen, um den Teleportationsprozess klarer zu kennzeichnen.
- **Fehlerbehandlung und Debugging:**
  - Teste die Teleportation gründlich, insbesondere wenn es um unvorhersehbare Bewegungen des Spielers geht. Implementiere Debugging-Hilfen, wie etwa eine visuelle Anzeige der Vektoren, die für die Berechnungen verwendet werden, um zu überprüfen, ob das Teleportationsverhalten korrekt funktioniert.
- **Interaktive Objekte im Portalbereich:**
  - Wenn sich interaktive Objekte im Bereich der Portale befinden, sollte deren Verhalten überprüft werden. Beispielsweise müssen bewegliche Objekte oder der Spieler bei der Teleportation korrekt positioniert und ausgerichtet werden, um Kollisionen oder Fehlverhalten zu verhindern.
- **Sicherstellen der Portalsicherheit:**
  - Vermeide unerwünschte Seiteneffekte wie z. B. das Eindringen des Spielers in Wände oder das Entstehen von "Löchern" in der Geometrie, indem du die Logik der Positionierung und Rotation der Spielerbewegung regelmäßig überprüfst.
- **Multiplayer-Überlegungen (falls zutreffend):**
  - In Multiplayer-Spielen müssen Portale sicherstellen, dass Teleportationen sowohl auf der Client- als auch auf der Serverseite synchronisiert werden, um Inkonsistenzen oder desynchronisierte Bewegungen zu vermeiden. Teste die Netzwerkübertragung der Portal-Positionen und der Spielerbewegungen gründlich.

Genutzte Tutorials und Assets:

- **Tutorial**:
  - Brackeys. (07.01.2018). Smooth PORTALS in Unity [Video]. YouTube. Verfügbar unter: https://youtu.be/cuQao3hEKfs?si=AzIBX-Jq7jEvdrL0
- **Assets**:
  - Brackeys. (01.05.2018). ScreenCutoutShader.shader [Shader]. GitHub. Verfügbar unter: https://github.com/Brackeys/Portal-In-Unity/blob/master/Portal/Assets/ScreenCutoutShader.shader
  - rpgwhiteloc*k.* (letzte Veröffentlichung: 20.07.2024). AllSky Free - 10 Sky / Skybox Set [Asset]. Unity Asset Store. Verfügbar unter:

https://assetstore.unity.com/packages/2d/textures-materials/sky/allsky-free-10-sky-skybox-set-146014
    - TriForge Assets. (letzte Veröffentlichung: 23.07.2019). Fantasy Forest Environment - Free Demo [Asset]. Unity Asset Store. Verfügbar unter: https://assetstore.unity.com/packages/3d/environments/fantasy/fantasy-forest-environment-free-demo-35361

Empfehlungen:

- Dani Krossing. (07.12.2021). HOW TO TELEPORT OBJECTS IN UNITY 🎮 | Teleport Player and GameObjects in Unity | Unity Tutorial [Video]. YouTube. Verfügbar unter: https://youtu.be/xmhm5jGwonc?si=idW07fOvMZVZNKKQ
- passivestar. (26.06.2022). Teleporting - Unity Tutorial [Video]. YouTube. Verfügbar unter: https://youtu.be/I7M8T3qU-_E?si=tlLdLHbhWKWbJjxt
- Daniel Ilett. (05.01.2022). Impossible Geometry with Stencil Shaders in Unity URP [Video]. YouTube. Verfügbar unter: https://youtu.be/EzM8LGzMjmc?si=TrL2_s_IrOSeYoyy

---

2.1.2 Dynamische und interaktive Raumveränderung

2.1.2.1 Dynamischer Raum (3)

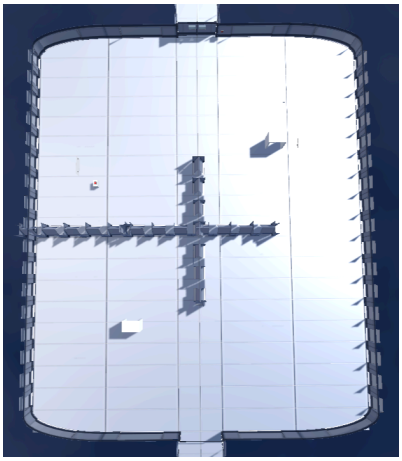

Abbildung 6: Raumaufbau

Umsetzung:

Raumstruktur:

- **Aufbau in zwei Hauptbereiche:**
    - **Start-Raum:**

        - Der Einstiegspunkt für den Spieler.
    - **Hauptraum:**
        - Verbunden mit dem Start-Raum über einen Gang; in der Mitte befindet sich eine kreuzförmige Wand, die den Raum in vier Abschnitte unterteilt. Diese Wand ist nicht mit den Außenwänden verbunden, sodass der Spieler frei im Kreis laufen kann.
- **Zusätzliche Räume:**
    - Auf der gegenüberliegenden Seite des Hauptraums führt ein weiterer Gang zu einem Raum, der dem Start-Raum gleicht.
- **Dynamischer Wechsel:**
    - Ein Durchgang ist zunächst verschlossen.
    - Durch einen Trigger im rechten Durchgang verschwindet eine Wand, sodass der Spieler rechts lang laufen kann.
    - Beim Zurücklaufen aktiviert ein zweiter Trigger, dass sich die rechte Wand schließt.

Wichtige Hinweise:

- **Raumfluss und Spielerführung:**
    - Überlege dir, wie der Spieler durch den Raum navigiert und wie sich der Raum dynamisch verändert, um den Fluss des Spiels zu steuern. Achte darauf, dass der Spieler jederzeit klare Hinweise bekommt, wohin er gehen soll, insbesondere bei der Aktivierung von dynamischen Raumänderungen. Verwende visuelle Hinweise wie Lichtquellen, Objekte oder Markierungen, um den Spieler zu leiten.
- **Verborgene und unerwartete Trigger:**
    - Experimentiere mit unsichtbaren Triggern, die durch bestimmte Aktionen oder Zeitverzögerungen aktiviert werden. Dadurch kannst du Überraschungsmomente schaffen, die der Spieler nicht sofort erwartet. Achte darauf, dass der Trigger-Mechanismus klar genug ist, um Verwirrung zu vermeiden, aber trotzdem überraschend bleibt.
- **Visuelle Rückmeldungen:**
    - Wenn sich ein Raum verändert (z. B. eine Wand verschwindet oder ein neuer Gang entsteht), stelle sicher, dass der Spieler visuelle Rückmeldungen bekommt. Dies könnte durch Animationen, Soundeffekte oder Veränderungen in der Beleuchtung erfolgen. Auf diese Weise wird dem Spieler klar, dass sich der Raum dynamisch verändert, ohne dass er explizit darauf hingewiesen werden muss.
- **Vermeidung von Verwirrung durch zu viele Veränderungen:**
    - Achte darauf, dass die Veränderungen des Raumes nicht zu häufig oder zu drastisch sind, da dies den Spieler desorientieren könnte. Dynamische Veränderungen sollten das Erlebnis bereichern, aber nicht die Übersichtlichkeit des Spiels beeinträchtigen. Eine Balance zwischen Überraschung und Klarheit ist entscheidend.

- **Trigger-Positionierung und Reihenfolge:**
    - Achte darauf, dass Trigger in einer logischen Reihenfolge und Position innerhalb des Raumes platziert sind, damit der Spieler intuitiv versteht, welche Aktionen er ausführen muss. Die Reihenfolge der Raumveränderungen sollte dem Spieler eine klare Vorstellung davon geben, welche Aufgaben er zu erfüllen hat.
- **Interaktive Elemente und Spielerfeedback:**
    - Plane, wie der Spieler mit den dynamischen Raumveränderungen interagieren kann. Wird er eine Wand aktivieren oder deaktivieren, indem er eine bestimmte Handlung vornimmt, oder passiert dies automatisch, wenn der Spieler in den Raum eintritt? Achte darauf, dass das Feedback zur Interaktion (z. B. durch Geräusche, Animationen oder Text) deutlich und verständlich ist.
- **Vermeidung von unnötigen Ladezeiten:**
    - Stelle sicher, dass dynamische Veränderungen des Raumes (wie das Erscheinen oder Verschwinden von Wänden) ohne merkliche Ladezeiten oder Ruckeln ablaufen. Ladezeiten können den Spielfluss stören und das Gefühl der Immersion brechen. Verwende Caching-Techniken oder Asynchronität, um Ladezeiten zu minimieren.
- **Testen der Spielererfahrung:**
    - Teste regelmäßig, wie der Spieler die dynamischen Veränderungen des Raums wahrnimmt und ob diese das Spielerlebnis verbessern oder verwirren. Es ist wichtig, dass der Spieler die Veränderungen nachvollziehen und als Teil des Spiels akzeptieren kann, ohne das Gefühl zu haben, dass sie willkürlich oder unangemessen sind.
- **Kombination von Triggern und Raumgestaltung:**
    - Stelle sicher, dass die Raumgestaltung und die Triggermechanismen so kombiniert werden, dass der Spieler nicht das Gefühl hat, dass die Veränderungen nur durch zufällige Platzierungen von Triggern passieren. Jede Veränderung sollte im Kontext des Raumes und der Spielsituation sinnvoll und nachvollziehbar erscheinen.
- **Kompatibilität mit verschiedenen Spielstilen:**
    - Überlege dir, wie unterschiedliche Spieler mit den dynamischen Änderungen des Raumes umgehen könnten. Einige Spieler bevorzugen vielleicht eine sehr lineare Erfahrung, während andere sich mehr in offenen, variablen Umgebungen zurechtfinden. Berücksichtige diese unterschiedlichen Spielstile bei der Gestaltung der dynamischen Raumveränderungen.

Genutzte Tutorials und Assets:

- **Assets**:
    - Barking Dog. (letzte Veröffentlichung: 22.07.2024). 3D Free Modular Kit [Asset]. Unity Asset Store. Verfügbar unter: https://assetstore.unity.com/packages/3d/environments/3d-free-modular-kit-85732

Empfehlungen:

- Karl Henkel. (12.03.2015). ProBuilder Basic Room Tutorial [Video]. YouTube. Verfügbar unter: https://youtu.be/ll7PpGXuHwI?si=nOhpILyrrrQ6udbS
- DarkDax. (28.02.2023). 12 Level Design Tips to Optimize Your Indie Game You Should Know [Video]. YouTube. Verfügbar unter: https://youtu.be/H-2IYnM3x8g?si=hlcasGimHH3P9LL4
- Brackyt. (09.02.2020). Room Changing Effect - Unity [Video]. YouTube. Verfügbar unter: https://youtu.be/CWNAGm36_gg?si=113sKloru5Pdl8mp

---

2.1.2.2 Trigger-Mechanik (2)

Unsichtbare Trigger:

- Erstelle ein Panel und füge einen Box Collider hinzu (wähle Trigger aus)
    - Entferne oder deaktiviere die MeshRenderer Komponente
- Hier kann nun ein Script angehängt werden (wie z.B. um Objekte zu Toggeln, siehe 2.1.2.3 Objekte Toggeln)

Sichtbare Trigger:

- Buttons oder Hebel bauen
    - Simpler Button: Erstelle ein Empty Game Objekt, das einen Zylinder für den Knopf und einen Würfel für die Box (optional)
    - Diese können ebenfalls animiert werden um die Interaktion zu verdeutlichen (z.B. Bewegung und Farbänderung)
- Hier kann nun ein Script angehängt werden (wie z.B. um Objekte zu Toggeln, siehe 2.1.2.3 Objekte Toggeln)

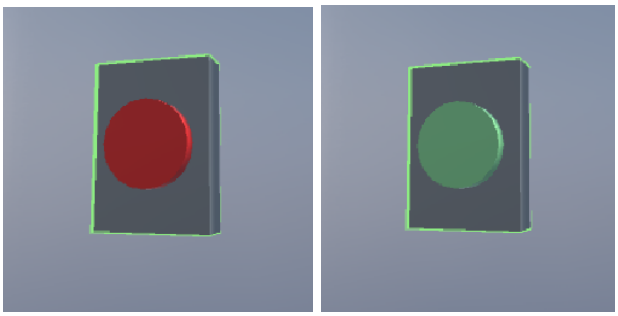

Abbildung 7: Trigger Buttons

Skript:

KnobAnimation.cs

Pfad: Assets → Scripts → Trigger → KnobAnimation.cs

```
using System.Collections;
using UnityEngine;

public class KnobAnimation : MonoBehaviour
{
    [Header("Animationseinstellungen")]
    [Tooltip("Wie weit der Knopf nach unten fährt (lokaler Abstand)")]
    public float pressDistance = 0.1f;

    [Tooltip("Dauer des Herunterfahrens (in Sekunden)")]
    public float pressDuration = 0.1f;

    private Vector3 initialLocalPosition;

    private void Start()
    {
        initialLocalPosition = transform.localPosition;
    }

    public void AnimatePress()
    {
        StartCoroutine(PressCoroutine());
    }

    private IEnumerator PressCoroutine()
    {
        Vector3 pressedPosition = initialLocalPosition - new Vector3(0,
        pressDistance, 0);

        float elapsed = 0f;
        while (elapsed < pressDuration)
        {
            transform.localPosition = Vector3.Lerp(initialLocalPosition,
            pressedPosition, elapsed / pressDuration);
            elapsed += Time.deltaTime;
            yield return null;
        }
        transform.localPosition = pressedPosition;

        elapsed = 0f;
        while (elapsed < pressDuration)
```

```
        {
            transform.localPosition = Vector3.Lerp(pressedPosition,
initialLocalPosition, elapsed / pressDuration);
            elapsed += Time.deltaTime;
            yield return null;
        }
        transform.localPosition = initialLocalPosition;
    }
}
```

AnimatePress:

- Startet die Coroutine *PressCoroutine()*, die den Drück-Animationseffekt ausführt.

Coroutine PressCoroutine:

- **Berechnung der gedrückten Position:**
  - Bestimmt *pressedPosition* als die ursprüngliche Position minus *pressDistance* entlang der Y-Achse.
- **Drücken-Phase:**
  - Nutzt eine Schleife, die über *pressDuration* läuft, und interpoliert die Position von *initialLocalPosition* zu *pressedPosition* mittels *Vector3.Lerp*.
  - Aktualisiert die Position in jedem Frame, bis die volle Drückdauer erreicht ist.
  - Setzt abschließend die Position exakt auf *pressedPosition*.
- **Zurückkehren-Phase:**
  - Startet eine weitere Schleife, die ebenfalls über *pressDuration* läuft, und interpoliert die Position von *pressedPosition* zurück zu *initialLocalPosition*.
  - Aktualisiert kontinuierlich die Position, bis der Knopf wieder in seiner Ausgangsposition steht.
  - Setzt abschließend die Position exakt auf *initialLocalPosition*.

Wichtige Hinweise:

- **Trigger-Aktivierung:**
  - Stelle sicher, dass Trigger nur dann aktiviert werden, wenn der Spieler wirklich in Kontakt mit dem Collider tritt. Vermeide es, Trigger durch ungewollte Kollisionen zu aktivieren, z. B. wenn der Spieler nur an der Nähe eines Objekts vorbeiläuft. Überprüfe, ob der Collider korrekt positioniert und dimensioniert ist, um eine präzise Interaktion zu gewährleisten.
- **Vermeidung von Trigger-Fehlern:**
  - Achte darauf, dass sich Triggerbereiche nicht versehentlich überschneiden, insbesondere bei unsichtbaren Triggern. Das könnte dazu führen, dass mehrere Trigger gleichzeitig aktiviert werden, was zu unerwünschten oder unerklärlichen

Ergebnissen führen kann. Teste alle Trigger gründlich in verschiedenen Szenarien, um solche Fehler zu vermeiden.

- **Visuelles Feedback für unsichtbare Trigger:**
  - Obwohl unsichtbare Trigger keine sichtbaren Objekte haben, kann es hilfreich sein, eine Art visuelles Feedback hinzuzufügen (z. B. durch Veränderung der Umgebung, wie das Flimmern eines Objekts oder einer leichten Veränderung der Lichtverhältnisse), damit der Spieler nicht aus Versehen an einem Trigger vorbeiläuft, ohne zu wissen, dass er in Interaktionsbereitschaft ist.
- **Trigger-Reaktionen optimieren:**
  - Wenn Trigger bestimmte Objekte aktivieren oder deaktivieren (wie das Toggeln von Objekten), achte darauf, dass diese Reaktionen sofort und ohne Verzögerung geschehen. Verzögerungen oder Ruckler können die Spielererfahrung negativ beeinflussen und den Spielfluss stören.
- **Interaktive Elemente optimieren:**
  - Wenn der Trigger mit interaktiven Elementen wie Buttons oder Hebeln verbunden ist, stelle sicher, dass die Animationen, wie z. B. das Drücken eines Knopfs, klar und flüssig sind. Wenn du visuelle Hinweise hinzufügst, wie z. B. das Hervorheben des Knopfs oder die Änderung der Farbe, könnte das dem Spieler helfen, den Fortschritt der Interaktion zu verstehen.
- **Anpassung der Trigger-Logik für unterschiedliche Spielsituationen:**
  - Überlege dir, wie Trigger in verschiedenen Spielsituationen und mit unterschiedlichen Spielstilen wirken können. Ein Trigger, der beim ersten Betreten eines Raums aktiviert wird, sollte nicht für den gesamten Spielverlauf verwendet werden. Stelle sicher, dass die Trigger-Logik gut zu den unterschiedlichen Anforderungen des Spiels passt, damit sie flexibel und sinnvoll genutzt werden kann.
- **Kombination von Triggern und Audio-/Visuelle Effekten:**
  - Setze Audio- oder visuelle Effekte ein, um die Aktivierung oder Deaktivierung eines Triggers zu begleiten. Ein Klickgeräusch oder eine visuelle Veränderung des Objekts kann die Interaktion klarer und ansprechender gestalten und zur Immersion des Spiels beitragen.
- **Unterscheidung zwischen verschiedenen Trigger-Typen:**
  - Unterscheide zwischen Triggern, die durch den Spieler aktiviert werden (z. B. Hebel oder Knöpfe), und Triggern, die automatisch durch das Spielgeschehen aktiviert werden (z. B. nach einer bestimmten Zeit oder bei einer bestimmten Spieleraktion). Plane unterschiedliche Trigger-Logiken für diese Fälle und berücksichtige, dass die Spieler möglicherweise unterschiedliche Erwartungen an diese Arten von Mechaniken haben.
- **Rückgängig machen von Trigger-Interaktionen:**
  - Überlege dir, ob und wie der Spieler Interaktionen mit einem Trigger rückgängig machen kann. Sollte ein Knopf z. B. nach dem Drücken wieder in die Ausgangsposition zurückkehren? Wenn ja, stelle sicher, dass die Rückkehranimation oder -logik ebenso reibungslos wie die Initial-Interaktion funktioniert.

- **Optimierung der Performance bei vielen Triggern:**
    - Wenn dein Spiel viele Trigger enthält, stelle sicher, dass die Performance des Spiels nicht negativ beeinflusst wird. Vermeide unnötige Berechnungen, indem du nur Trigger überprüfst, die sich im Bereich des Spielers oder seiner Nähe befinden. Dies kann helfen, die Leistung in größeren Levels oder bei intensiven Interaktionen zu optimieren.

2.1.2.3 Objekte Toggeln (1)

Umsetzung:

- Füge dem Triggern das Skript hinzu, der Objekte aktiviert und deaktiviert

---

Skript:

ToggleObjectWithCount.cs

Pfad: Assets → Scripts → SpacialIllusion → ToggleObjectWithCount.cs

```
using UnityEngine;

public class ToggleObjectWithCount : MonoBehaviour
{
    [Header("Zu toggelndes Objekt")]

    public GameObject targetObject;

    [Header("Anzahl Durchläufe bis Toggle")]

    public int requiredTriggerCount = 3;
    private int triggerCounter = 0;

    private void OnTriggerEnter(Collider other)
    {

        if (other.CompareTag("Player"))
        {
            triggerCounter++;
            if (triggerCounter >= requiredTriggerCount)
            {
                targetObject.SetActive(!targetObject.activeSelf);
```

```
                triggerCounter = 0;
            }
        }
    }
}
```

Trigger-Überprüfung:

- In der Methode OnTriggerEnter(Collider other) wird geprüft, ob das Objekt, das den Trigger betritt, den Tag "Player" besitzt.

Zähler erhöhen:

- Bei einem Treffer des Spielers wird der Zähler triggerCounter um eins erhöht.

Toggle-Logik:

- Wenn der Zähler den Wert von requiredTriggerCount erreicht oder überschreitet, wird:
    - Der Aktivitätsstatus des targetObject umgekehrt (ein-/ausgeschaltet).
    - Der Zähler triggerCounter wieder auf 0 gesetzt, um einen neuen Zyklus zu starten.

Wichtige Hinweise:

- **Performance bei wiederholtem Triggern:**
    - Achte darauf, dass das wiederholte Aktivieren und Deaktivieren von Objekten keine Leistungseinbußen verursacht, insbesondere wenn du viele Objekte toggeln lässt. Wenn der Trigger häufig ausgelöst wird, kann es hilfreich sein, die Zähler-Logik zu optimieren oder unnötige Überprüfungen zu vermeiden.
- **Feedback an den Spieler:**
    - Gib dem Spieler visuelles oder akustisches Feedback, wenn ein Objekt durch den Trigger aktiviert oder deaktiviert wird. Dies kann dazu beitragen, dass der Spieler besser versteht, welche Mechanik im Spiel aktiv ist, und die Interaktion klarer und intuitiver wird.
- **Visuelle Indikatoren für den Trigger:**
    - Stelle sicher, dass der Spieler visuell erkennen kann, wann der Zähler für das Toggeln aktiv wird. Dies könnte durch Änderungen an der Farbe oder dem Zustand des Triggers geschehen, wenn der Zähler die festgelegte Anzahl erreicht hat, sodass der Spieler genau weiß, wann eine Interaktion erfolgreich war.
- **Vermeidung von unbeabsichtigtem mehrfachen Toggeln:**
    - Es sollte verhindert werden, dass der Spieler durch wiederholtes Betreten des Triggers unbeabsichtigt zu viele Toggeln in kurzer Zeit auslöst. Stelle sicher, dass

der Zähler nach jedem erfolgreichen Toggeln zurückgesetzt wird, und bedenke, ob eine zusätzliche Verzögerung zwischen den Trigger-Aktivierungen erforderlich ist, um zu vermeiden, dass der Spieler ungewollt mehrere Toggeln hintereinander auslöst.

- **Trigger-Überprüfung und Kollisionserkennung:**
  - Achte darauf, dass die Kollisionserkennung zwischen dem Spieler und dem Trigger korrekt funktioniert. Wenn du mit verschiedenen Triggerarten arbeitest (z. B. unsichtbare Trigger oder Trigger, die nur eine bestimmte Zeit lang aktiv sind), stelle sicher, dass der OnTriggerEnter-Mechanismus ordnungsgemäß ausgelöst wird, ohne falsche Kollisionen oder ungewollte Interaktionen zu verursachen.
- **Verhalten bei deaktivierten Objekten:**
  - Überlege, wie das Verhalten des Spiels aussieht, wenn das getogglete Objekt deaktiviert wird. Wird es in seiner Deaktivierung statisch oder kann der Spieler weiterhin mit ihm interagieren, auch wenn es unsichtbar oder deaktiviert ist? Achte darauf, dass das deaktivierte Objekt sich nicht unvorhersehbar verhält.
- **Animationen und Übergänge beim Toggeln:**
  - Wenn das toggelnde Objekt animiert ist, stelle sicher, dass die Animationen beim Aktivieren und Deaktivieren korrekt ablaufen. Eine unterbrochene oder fehlerhafte Animation kann zu Verwirrung führen und das Spielerlebnis negativ beeinflussen. Überprüfe, ob die Animation sanft und ohne Ruckler abläuft.
- **Multiplayer-Kompatibilität:**
  - Wenn das Spiel Multiplayer-Funktionen hat, berücksichtige, wie das Toggeln von Objekten zwischen den Spielern synchronisiert wird. Du musst möglicherweise sicherstellen, dass der Trigger-Status korrekt zwischen allen Spielern geteilt wird, um Inkonsistenzen im Gameplay zu vermeiden.
- **Erweiterbarkeit:**
  - Die ToggleObjectWithCount.cs-Logik ist leicht erweiterbar, wenn du später zusätzliche Funktionen hinzuzufügen möchtest, wie z. B. das Hinzufügen von Zeitverzögerungen zwischen den Toggelns oder das Hinzufügen von unterschiedlichen Effekten für verschiedene Objekte, ist dies möglich. Halte deinen Code modular und flexibel, um Änderungen in der Zukunft zu erleichtern.

2.1.2.4 Objekte Bewegen (2)

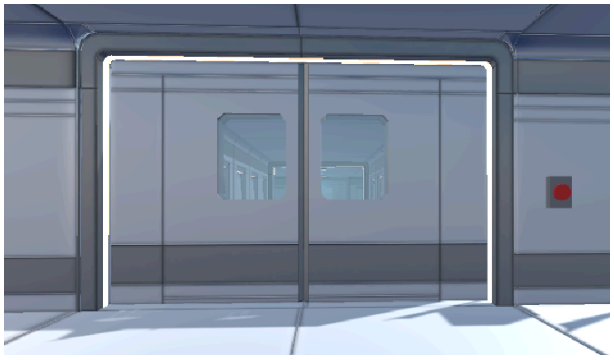

Abbildung 8: Trigger gesteuerte Tür

Umsetzung:

Türen:

- Tür Objekte erstellen aus dem Modular Kit in die Szene ziehen
    - Alternativ können zwei simple Würfel erstellt und skaliert werden

---

Skript:

Doorcontroller.cs

Pfad: Assets → Scripts → Trigger→ → Doorcontroller.cs

```
using UnityEngine;
using System.Collections;

public class DoorController : MonoBehaviour
{
    [Header("Tür Objekte")]
    public GameObject leftDoor;
    public GameObject rightDoor;

    [Header("Bewegungseinstellungen")]
    public float doorMoveDistance = 2.0f;
    public float moveDuration = 1.0f;

    [Header("Knopf Einstellungen (optional)")]
    public GameObject knob;
```

```
    private bool doorsOpen = false;
    private Vector3 leftDoorClosedPos;
    private Vector3 rightDoorClosedPos;

    void Start()
    {
        if (leftDoor != null)
            leftDoorClosedPos = leftDoor.transform.localPosition;
        if (rightDoor != null)
            rightDoorClosedPos = rightDoor.transform.localPosition;
    }

    public void Interact()
    {
        if (!doorsOpen)
        {
            StartCoroutine(MoveDoor(leftDoor,
                leftDoor.transform.localPosition,
                leftDoorClosedPos + Vector3.left * doorMoveDistance,
                moveDuration));
                StartCoroutine(MoveDoor(rightDoor,
                rightDoor.transform.localPosition,
                rightDoorClosedPos + Vector3.right * doorMoveDistance,
                moveDuration));
            doorsOpen = true;
        }
        else
        {
            StartCoroutine(MoveDoor(leftDoor,
                leftDoor.transform.localPosition,
                leftDoorClosedPos,
                moveDuration));
            StartCoroutine(MoveDoor(rightDoor,
                rightDoor.transform.localPosition,
                rightDoorClosedPos,
                moveDuration));
            doorsOpen = false;
        }

        if (knob != null)
        {
            KnobColorToggle kct = knob.GetComponent<KnobColorToggle>();
            if (kct != null)
```

```
                kct.ToggleColor();

            KnobAnimation ka = knob.GetComponent<KnobAnimation>();
            if (ka != null)
                ka.AnimatePress();
        }
    }

    IEnumerator MoveDoor(GameObject door, Vector3 startPos, Vector3 endPos,
    float duration)
    {
        if (door == null)
            yield break;

        float elapsed = 0f;
        while (elapsed < duration)
        {
            door.transform.localPosition = Vector3.Lerp(startPos, endPos,
            elapsed / duration);
            elapsed += Time.deltaTime;
            yield return null;
        }
        door.transform.localPosition = endPos;
    }
}
```

Interaktionsmethode:

- **Türzustandsprüfung:**
  - Überprüft, ob die Türen momentan geschlossen oder geöffnet sind (boolescher Zustand doorsOpen).
- **Türbewegung:**
  - **Öffnen:**
    - Startet Coroutinen, die die linke Tür nach links und die rechte Tür nach rechts bewegen, basierend auf doorMoveDistance.
    - Setzt doorsOpen auf true.
  - **Schließen:**
    - Startet Coroutinen, die die Türen zurück zu ihren ursprünglichen Positionen bewegen.
    - Setzt doorsOpen auf false.
- **Knopf-Interaktion (optional):**
  - Falls ein Knopf vorhanden ist, werden zwei Komponenten geprüft:

        - KnobColorToggle: Wechselt die Farbe des Knopfes.
        - KnobAnimation: Löst eine Animationssequenz (z. B. visuelle Bestätigung eines Tastendrucks) aus.

Türbewegung über Coroutine:

- **Parameter:**
    - Empfängt das zu bewegende Türobjekt, Startposition, Endposition und die Dauer der Bewegung.
- **Bewegungsablauf:**
    - Nutzt eine Schleife, die über die angegebene *duration* läuft.
    - Berechnet mittels *Vector3.Lerp* den interpolierten Wert zwischen Start- und Endposition basierend auf der vergangenen Zeit.
    - Aktualisiert kontinuierlich die Position der Tür, bis die volle Dauer erreicht ist.
    - Setzt abschließend die Position exakt auf die Endposition.

Wichtige Hinweise:

- **Türbewegung bei komplexeren Interaktionen:**
    - Wenn Türen als Teil komplexerer Interaktionen (z. B. in einem Puzzle) verwendet werden, achte darauf, dass der Mechanismus gut in den Gesamtfluss des Spiels integriert wird. Mehrere Türen, die gleichzeitig geöffnet oder geschlossen werden, können zu Problemen führen, wenn der Spieler nicht erwartet, dass alle Türen gleichzeitig reagieren. Es könnte sinnvoll sein, eine zentrale Steuerung für alle Türen einzuführen, die ihr Verhalten steuert.
- **Visuelle und akustische Effekte:**
    - Stelle sicher, dass die Bewegung der Türen von klaren visuellen und akustischen Effekten begleitet wird. Eine einfache Animation reicht oft nicht aus, um die Interaktivität klar zu machen. Eine Tür, die sich öffnet oder schließt, könnte von einem Geräusch oder einer Bewegungseffekt-Animation begleitet werden, um dem Spieler ein besseres Gefühl für die Interaktion zu geben.
- **Performance-Optimierung bei mehreren Türen:**
    - Wenn du viele Türen in einer Szene hast, könnte es zu Performance-Problemen kommen, besonders wenn alle Türen regelmäßig bewegt werden. Überlege, ob es sinnvoll ist, Türen nur dann zu animieren, wenn sie sich tatsächlich in der Nähe des Spielers oder der Kamera befinden. Das verhindert unnötige Berechnungen und verbessert die Performance.
- **Zustandskontrolle und Synchronisierung:**
    - Achte darauf, dass der Türzustand (offen oder geschlossen) korrekt überwacht wird. Wenn eine Tür durch einen Trigger geöffnet wird und der Spieler die Szene verlässt, während die Tür noch geöffnet ist, könnte es zu Inkonsistenzen kommen. Verwende daher eine Art "Zustandsmanager", der sicherstellt, dass

Türen auch nach Szenenwechseln oder Wiederholungen korrekt verwaltet werden.

- **Vermeidung von Kollisionen und Blockierungen:**
  - Überprüfe, dass die Türen bei der Bewegung nicht mit anderen Objekten kollidieren oder diese blockieren. Insbesondere wenn Türen zur Spielumgebung gehören, die auch andere Interaktionen oder Bewegungen von Objekten erfordern, kann es hilfreich sein, Kollisionen zu deaktivieren, während sich die Tür bewegt.
- **Mehrere Interaktionsmöglichkeiten:**
  - Wenn du nicht nur einen Knopf als Interaktionsmittel verwendest, sondern auch andere Mechaniken (z. B. Druckplatten oder Lichttrigger), stelle sicher, dass die Tür nur auf die beabsichtigte Art und Weise reagiert. Mische nicht verschiedene Arten von Interaktionen zu einem Trigger, um unklare Ergebnisse zu vermeiden.
- **Sichtbarkeit und Zugänglichkeit des Knopfes:**
  - Stelle sicher, dass der Knopf, wenn er als Interaktionsobjekt dient, immer in einer klaren und sichtbaren Position für den Spieler platziert wird. Vermeide es, ihn in schwer zugänglichen Bereichen zu verstecken, da dies das Gameplay unnötig erschwert.
- **Fehlerbehebung bei Bewegungsproblemen:**
  - Wenn du Probleme mit der Türbewegung hast (z. B. die Türen bleiben in der Mitte hängen oder bewegen sich nicht richtig), überprüfe alle relevanten Parameter, wie doorMoveDistance und moveDuration. Auch die Positionen der Türen sollten regelmäßig überprüft werden, um sicherzustellen, dass keine unerwünschten Verschiebungen auftreten.
- **Interaktive Animationen:**
  - Wenn der Knopf eine visuelle Animation hat, wie z. B. das Herunterdrücken, stelle sicher, dass diese Animation während der Türinteraktion richtig synchronisiert wird. Falls es einen Knopf gibt, der gleichzeitig eine Türbewegung auslöst, sollte die Animation des Knopfes nicht überstürzt oder unterbrochen werden, um das Benutzererlebnis zu optimieren.
- **Testing bei verschiedenen Kameraperspektiven:**
  - Teste die Bewegungen und Interaktionen aus verschiedenen Kamerawinkeln. Was auf einer festen Kamera gut funktioniert, könnte bei einer freiere Kameraansicht oder im First-Person-Modus fehlerhaft wirken, wenn der Spieler z. B. zu nahe an der Tür steht oder die Bewegung der Tür blockiert wird.
- **Kompatibilität mit VR oder First-Person-Perspektiven:**
  - Falls du VR oder First-Person-Gameplay berücksichtigst, stelle sicher, dass die Türbewegung für diese Perspektiven angepasst wird. Eine Tür, die sich nur auf einer bestimmten Achse bewegt, könnte aus einer nahen Perspektive seltsam wirken. Stelle sicher, dass die Bewegungsrichtung der Türen immer intuitiv und an die Perspektive des Spielers angepasst ist.
- **Schutz vor ungewolltem Spamming der Interaktion:**
  - Wenn der Spieler wiederholt gegen den Knopf oder den Trigger läuft, könnte das zu ungewolltem Öffnen oder Schließen der Türen führen. Implementiere ein

Cooldown oder eine Verzögerung, die verhindert, dass die Tür zu schnell nach einer Interaktion wieder geöffnet oder geschlossen wird.

Genutzte Tutorials und Assets:

- **Tutorial**:
  - Jayanam. (05.05.2018). Unity Beginner Tutorial : Triggers [Video]. YouTube. Verfügbar unter: https://youtu.be/JHZN_vMqqr4?si=MT6je6KKcSTtlLvj
- **Assets**:
  - Barking Dog. (letzte Veröffentlichung: 22.07.2024). 3D Free Modular Kit [Asset]. Unity Asset Store. Verfügbar unter: https://assetstore.unity.com/packages/3d/environments/3d-free-modular-kit-85732

Empfehlungen:

- LlamAcademy. (09.11.2021). Doors in Unity - Sliding and Rotating Doors | Make Them Automatically Open/Close or Usable [Video]. YouTube. Verfügbar unter: https://youtu.be/cPltQK5LIGE?si=2LeFOMMqxV-A_9IZ

2.1.3 Räumliche Tiefe (1)

Abbildung 9: Tiefenwirkung

Umsetzung:

Scene:

- **Lighting settings**:
  - Unter Window → Rendering → Lighting → Scene → Lighting Settings As
    - setze die Lighting Settings auf Cold SunsetSettings

Environment:

- **Environment:**
  - Unter Window → Rendering → Lighting → Environment→ Environment→ Skybox Material
    - setze das Skybox Material auf Cold Sunset

Other Settings:

- **Fog:**
  - Unter Window → Rendering → Lighting → Environment→ Other Settings→ Fog
    - wähle die Checkbox aus
    - setze Color auf eine helle blaue Farbe (z.B. Hexadezimal: ABF1EC)
    - setze Density auf 0.02

Wichtige Hinweise:

- **Verwendung von Lichtquellen zur Tiefe:**
  - Experimentiere mit Lichtquellen, um die räumliche Tiefe weiter zu verstärken. Nutze z. B. gerichtetes Licht, um die Illusion von Entfernungen zu erzeugen. Achte darauf, dass Schatten korrekt geworfen werden, da diese die Wahrnehmung der Tiefe erheblich beeinflussen. Verwende gegebenenfalls auch Lichtquellen mit unterschiedlicher Intensität und Farbe, um die Atmosphäre und Tiefe zu variieren.
- **Lichtstärke und -farben anpassen:**
  - Variiere die Lichtstärke und -farben in verschiedenen Bereichen der Szene. Helle, warme Farben können den Vordergrund betonen, während kühle und dunklere Farbtöne im Hintergrund die Illusion von Tiefe verstärken. Achte darauf, dass die Lichtquelle und deren Intensität in das gewünschte Ambiente und die Stimmung des Spiels passen.
- **Fog in Verbindung mit Licht:**
  - Fog ist ein effektives Mittel zur Schaffung von Tiefe, besonders in Umgebungen mit langen Sichtlinien. Nutze Fog zusammen mit Lichtquellen, um Objekte in der Ferne weicher und verschwommener erscheinen zu lassen, was der Szene eine

stärkere räumliche Tiefe verleiht. Achte darauf, die Dichte des Fog nicht zu hoch zu setzen, da dies die Sichtbarkeit beeinträchtigen könnte.

- **Fog und Partikeleffekte kombinieren:**
  - Kombiniere Fog mit Partikeleffekten (wie Staub, Rauch oder Nebel), um der Szene zusätzliche visuelle Tiefe und Realismus zu verleihen. Diese Effekte können durch die Lichtquellen beeinflusst werden und eine glaubwürdige Atmosphäre schaffen, die den Spieler tiefer in die Spielwelt eintauchen lässt.
- **Skybox und Nebel kombinieren:**
  - Stelle sicher, dass die Skybox mit dem Fog-Effekt harmoniert. Eine Skybox, die weit entfernte Berge oder Horizonte darstellt, kann durch Fog-Effekte noch realistischer wirken, da Objekte in der Ferne zunehmend weniger sichtbar werden. Dies verstärkt den Eindruck von räumlicher Tiefe.
- **Post-Processing für Tiefenunschärfe:**
  - Verwende Tiefenunschärfe (Depth of Field) als Post-Processing-Effekt, um Objekte im Hintergrund unscharf darzustellen. Dies lenkt den Fokus des Spielers auf den Vordergrund und verstärkt den Tiefeneindruck. Achte darauf, die Stärke der Unschärfe so anzupassen, dass sie nicht zu stark ist und die Lesbarkeit der Szene beeinträchtigt.
- **Verwenden von Layern für Tiefensteuerung:**
  - Ordne Objekte verschiedenen Layern zu, die unterschiedliche Tiefenwirkungen haben. Objekte, die weiter entfernt sind, können in weniger Detail und mit weniger Beleuchtung dargestellt werden. Dies unterstützt die Wahrnehmung von Entfernungen und räumlicher Tiefe.
- **Spezifische Lichtquellen für den Vordergrund:**
  - Setze zusätzlich zu den allgemeinen Umgebungslichtern punktuelle Lichtquellen im Vordergrund ein, um Objekte hervorzuheben und die Aufmerksamkeit des Spielers zu lenken. Diese Lichtquellen sollten mit den Hintergrundlichtern kontrastieren, um eine klarere Trennung zwischen Vorder- und Hintergrund zu erzeugen.
- **Skybox-Texturen und ihre Auswirkungen:**
  - Wähle eine Skybox, die das gewünschte Gefühl von Tiefe verstärkt. Eine detaillierte, texturierte Skybox kann helfen, das Gefühl einer weiten, offenen Welt zu vermitteln, während eine minimalistische oder stilisierte Skybox eher eine fokussierte und enge Atmosphäre schafft.
- **Vermeidung von visuellen Unstimmigkeiten:**
  - Achte darauf, dass der Fog-Effekt nicht zu plötzlich oder inkonsistent erscheint. Wenn der Übergang zwischen klarer Sicht und nebligem Bereich zu abrupt ist, könnte dies die visuelle Kohärenz der Szene stören. Experimentiere mit verschiedenen Übergangsbereichen, um einen natürlichen Effekt zu erzielen.
- **Interaktive Objekte und räumliche Wahrnehmung:**
  - Achte darauf, dass interaktive Objekte in der Szene klar aus der Tiefe hervortreten. Eine zu starke Integration von interaktiven Objekten in entfernte Bereiche könnte dazu führen, dass sie schwer wahrnehmbar werden. Verwende

Licht, Farbe und Bewegung, um die Interaktivität dieser Objekte zu betonen und die räumliche Wahrnehmung zu fördern.

- **Testen aus verschiedenen Perspektiven:**
    - Teste die Szene aus verschiedenen Kameraansichten und -abständen, um sicherzustellen, dass die erzeugte räumliche Tiefe auch bei unterschiedlichen Blickwinkeln gut funktioniert. Verändere beispielsweise die Kamerahöhe und den Winkel, um sicherzustellen, dass die Illusion der Tiefe nicht verloren geht.
- **Räumliche Tiefe in offenen Welten:**
    - In offenen Welten, in denen der Spieler eine weite Sichtweite hat, kann die Nutzung von Fog und einer geeigneten Skybox helfen, den Übergang von nahen zu entfernten Bereichen sanfter und natürlicher zu gestalten. Hierbei sollte der Fog so eingestellt werden, dass er sich mit der Entfernung der Objekte im Hintergrund verstärkt und somit eine glaubwürdige Tiefe erzeugt.

Empfehlungen:

- Brackeys.(24.01.2018). LIGHTING in Unity [Video]. YouTube. Verfügbar unter: https://youtu.be/VnG2gOKV9dw?si=AwERNk2fmTYIPDyT
- Fist Full of Shrimp. (05.01.2024). Light Up Your Game : Unity Lighting Essentials! [Video]. YouTube. Verfügbar unter: https://youtu.be/5rxMdiCkQGk?si=ULPkHYzYQ6RezQqf
- LMHPOLY. (11.11.2020). Unity 2020 Tutorial - Lighting And Post-Processing Low Poly Scene [Video]. YouTube. Verfügbar unter: https://youtu.be/uwYf1yMPuvk?si=jGD1-BiS0WaKANqm
- GDT Solutions ES. (16.11.2020). How to replace DEFAULT SKYBOX in Unity - Custom Skybox with 360 TEXTURE [Video]. YouTube. Verfügbar unter: https://youtu.be/ckgfQdaEdwk?si=8k9HI1A3_QWbEI_C
- Unity Ace. (04.01.2022). How to Add Fog in Unity [Video]. YouTube. Verfügbar unter: https://youtu.be/RBI9hQWD_xY?si=Tze9IXUacXnWTydm

---

2.4 Allgemeine Tipps & Warnungen

Regelmäßiges Testen:

- **Überprüfung der Trigger:** Teste kontinuierlich, ob alle Trigger reibungslos funktionieren und keine unerwünschten Übergänge auftreten. Dies ist entscheidend, um sicherzustellen, dass die Illusionen unter allen Bedingungen korrekt arbeiten.
- **Blickwinkel berücksichtigen:** Da räumliche Illusionen stark vom Blickwinkel des Spielers abhängen, sollte regelmäßig aus unterschiedlichen Perspektiven getestet werden, ob die Illusion stabil bleibt und nicht durch unerwünschte Sichtwinkel zerstört wird.

Feinabstimmung:

- **Licht- und Nebeleffekte:** Die Platzierung von Lichtquellen und Nebeleffekten kann die Wahrnehmung der Illusion verstärken oder schwächen. Achte darauf, diese Elemente präzise anzupassen, um die gewünschte Wirkung zu erzielen.
- **Optimale Objektplatzierung:** Experimentiere mit der Platzierung von Objekten, um zu prüfen, welche Anordnung die Illusion am besten unterstützt. Kleine Verschiebungen können große Unterschiede in der Wahrnehmung bewirken.

Performance beachten:

- **Rechenintensive Effekte:** Achte darauf, dass dynamische Effekte wie die Aktivierung und Deaktivierung von Objekten oder Veränderungen der Raumstruktur die Performance nicht negativ beeinflussen. Optimiere komplexe Mechaniken, um einen flüssigen Ablauf zu gewährleisten.
- **Profiler-Tools nutzen:** Verwende Profiling-Tools, um mögliche Engpässe bei der Performance frühzeitig zu identifizieren. Dies hilft, die Performance zu optimieren und sicherzustellen, dass die Illusionen reibungslos laufen.

Dokumentation und Debugging:

- **Logik und Hierarchie dokumentieren:** Dokumentiere die Logik der Trigger sowie die Objekt-Hierarchie, um spätere Optimierungen und Wartungsarbeiten zu erleichtern. Eine klare Struktur sorgt für besseren Überblick und schnellere Anpassungen.
- **Fehlerquellen erkennen:** Nutze Debugging-Tools, um unerwünschte Zustände und fehlerhafte Übergänge schnell zu identifizieren und zu beheben. Dies hilft, unerwartete Verhaltensweisen zu vermeiden.

Konsistenz in der Benutzererfahrung:

- **Nahtlose Übergänge:** Achte darauf, dass Übergänge zwischen verschiedenen Illusionen fließend und ohne Abrüche von der Wahrnehmung stattfinden. Abrupte Veränderungen könnten die Immersion des Spielers stören.
- **Visuelles Feedback:** Verwende visuelle Effekte, um subtile Hinweise zu geben, wie etwa durch Animationen oder Soundeffekte. Dies hilft dem Spieler, sich in der Illusion zurechtzufinden und die Welt immersiv zu erleben

---

2.5 Verwendungszweck

Erweiterte Immersion und Atmosphäre:

- Räumliche Illusionen ermöglichen es, den Spieler in eine komplexe und sich ständig verändernde Umgebung einzutauchen. Durch das Spiel mit der Raumwahrnehmung wird ein kontinuierliches Gefühl von Unvorhersehbarkeit und Spannung erzeugt, was die emotionale Bindung zum Spiel verstärkt.

Innovative Gameplay-Mechaniken:

- Räumliche Illusionen stellen klassische Raumkonzepte infrage und nutzen den Raum selbst als aktives Spielelement. Durch Mechaniken wie nicht-euklidische Portale können Spieler neue Wege entdecken und auf kreative Weise Rätsel lösen, was das traditionelle Raumverständnis erweitert.

Spannungssteigerung in Horror- und Mystery-Spielen:

- In Horror- oder Mystery-Spielen werden räumliche Illusionen verwendet, um die Umgebung dynamisch zu verändern und so ein Gefühl der Desorientierung und Bedrohung zu erzeugen. Die Manipulation der Raumstruktur verstärkt das Gefühl von Unsicherheit und steigert die Spannung, was besonders in psychologischen Horrorspielen zur Anwendung kommt.

Unterstützung narrativer Elemente:

- Räumliche Illusionen können auch eine narrative Rolle spielen, indem sie die Story visuell unterstützen. Durch dynamische Veränderungen der Umgebung lassen sich wichtige Wendepunkte und symbolische Darstellungen verstärken, was die Erzählung lebendiger und intensiver macht.

3 Größen Illusionen

3.1 Technische Implementierung

3.1.1 Proportion & Positionierung

3.1.1.1 Ebbinghaus-Illusion (1)

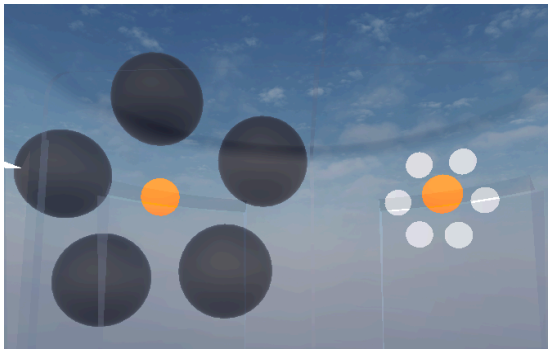

Abbildung 10: Ebbinghaus-Illusion

Umsetzung:

Zentren:

- Erstelle zwei gleichgroße Kugeln für das zentrum der Gebilde
    - positioniere die Kugeln nebeneinander

Schrumpfeffekt:

- Erstelle sechs weitere Kugeln
    - Umgebe das erste Zentrum mit kleineren Kugeln

Vergrößerungseffekt:

- Erstelle fünf weitere Kugeln
    - Umgebe das zweite Zentrum mit größeren Kugeln

Wichtige Hinweise:

- **Position und Abstände beachten:**
    - Stelle sicher, dass alle Kugeln in gleicher Höhe und in derselben Entfernung zum Betrachter positioniert sind. Variationen in der Entfernung und Höhe können die Wahrnehmung der Illusion beeinflussen und die Wirkung verfälschen.
- **Skalierung und Perspektive:**
    - Achte darauf, dass die Skalierung der Kugeln nicht nur proportional, sondern auch perspektivisch korrekt ist. Besonders bei variierenden Abständen der Kugeln zum Betrachter kann die Illusion durch falsche Skalierung des Objekts verzerrt werden.
- **Lichtverhältnisse:**
    - Nutze Licht und Schatten gezielt, um die Unterschiede in der Wahrnehmung von Größe und Abständen zu verstärken. Eine gezielte Beleuchtung der Kugeln kann dazu beitragen, den Effekt noch intensiver wirken zu lassen, da Schatten und Reflektionen die Größenwahrnehmung beeinflussen.
- **Farbwahl:**
    - Die Farbwahl der Kugeln kann ebenfalls einen Einfluss auf die Wahrnehmung haben. Achte darauf, dass die Kugeln unterschiedliche Farben haben, wenn du den Effekt verstärken möchtest. Helle Farben lassen Objekte oft größer erscheinen, während dunklere Farben sie kleiner wirken lassen können.
- **Kamerawinkel und Sichtfeld:**
    - Der Effekt der Ebbinghaus-Illusion kann je nach Kamerawinkel variieren. Teste den Effekt aus verschiedenen Perspektiven, um sicherzustellen, dass er aus der vorgesehenen Ansicht gut funktioniert.
- **Psychologische Komponente:**
    - Der Effekt hängt auch von der Wahrnehmung des Spielers ab, daher ist es wichtig, das Design nicht zu überladen. Zu viele Variationen in den Objekten können die Illusion schwächen, da der Spieler möglicherweise beginnt, die Größenunterschiede bewusst zu hinterfragen.
- **Vermeidung von Verzerrungen durch Kollision:**
    - Wenn die Kugeln in der Szene physische Interaktionen oder Kollisionen haben, achte darauf, dass die Kollisionen die Illusion nicht zerstören. Besonders bei größeren Objekten, die stark vergrößert oder verkleinert sind, könnte es zu unerwünschten Kollisionseffekten kommen. Teste daher die Interaktionen gründlich, um sicherzustellen, dass keine Verzerrungen auftreten.
- **Feedback zur Wahrnehmung:**
    - Ergänze die Illusion mit subtilen Animationen oder visuellen Hinweisen, die den Spieler darauf aufmerksam machen, dass die Größenänderung stattfindet. Dies

könnte die Wahrnehmung der Illusion verstärken und ein bewussteres Spielerlebnis schaffen.

3.1.1.2 Bewusste Positionierung (2)

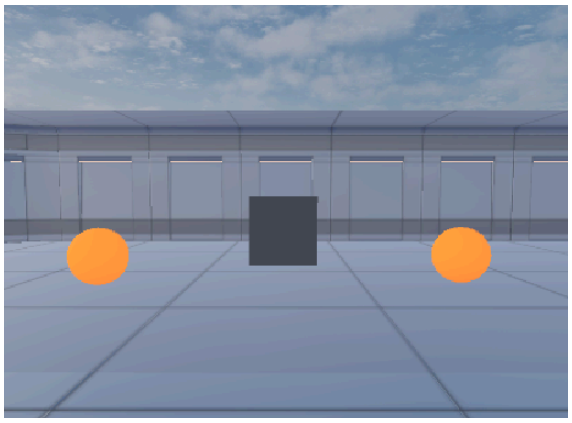

Abbildung 11: Größen Illusion durch Positionierung

Umsetzung:

Position:

- Lege die Position aus der die Szenen betrachtet werden sollen fest und markiere diese gegebenenfalls

Objekte:

- Erstelle zwei gleich große Würfel und platziere sie gegenüber voneinander

Schrumpfeffekt:

- Positioniere den Würfel auf dem Boden
- Erstelle zwei kleine Kugeln
    - Setze diese jeweils rechts und links neben den ersten Würfel

Vergrößerungseffekt:

- Positioniere den Würfel höher als den gegenüberliegenden Würfel (Y-Achse)
    - setze die Entfernung basierend auf den gegenüberliegenden Würfel

        - beide Würfel sollen so wirken als wären sie in derselben Entfernung zum Betrachter
- Erstelle zwei große Kugeln
    - Setze diese jeweils rechts und links neben den zweiten Würfel

---

Wichtige Hinweise:

- **Betrachterposition und Blickwinkel:**
    - Stelle sicher, dass die Position des Betrachters für den Effekt korrekt festgelegt ist. Kleine Änderungen in der Kameraausrichtung können den Effekt der Größenwahrnehmung stark beeinflussen. Achte darauf, dass der Spieler immer aus der gewünschten Perspektive schaut, um die Täuschung zu verstärken.
- **Abstände und Perspektive:**
    - Auch wenn beide Würfel gleich groß sind, sollte der Abstand zwischen den Objekten und ihre Position zueinander so gewählt werden, dass der zweite Würfel durch den höheren Stand und die damit verbundene Entfernung kleiner erscheint. Um dies zu erzielen, kann eine gezielte Steuerung der Z-Achse (Tiefe) in der Szene nötig sein.
- **Eindeutige Positionierung der Kugeln:**
    - Die Position der Kugeln spielt eine zentrale Rolle bei der Wahrnehmung der Größenillusion. Stelle sicher, dass sie immer gleichmäßig links und rechts der Würfel positioniert sind. Die Größenwahrnehmung kann stark von der Symmetrie beeinflusst werden, also achte auf eine genaue Platzierung.
- **Korrekte Verwendung der Y-Achse:**
    - Beim höheren Platzieren des zweiten Würfels ist es wichtig, den Effekt der vertikalen Position korrekt zu nutzen. Das Verändern der Höhe sollte in Kombination mit der Perspektive genutzt werden, um die Illusion der Größenveränderung zu verstärken. Der Würfel muss so platziert werden, dass der Blickwinkel auf den Abstand und die Höhe den Schrumpf- oder Vergrößerungseffekt begünstigt.
- **Licht und Schatten berücksichtigen:**
    - Lichtverhältnisse können die Wahrnehmung der Position und Größe von Objekten beeinflussen. Ein heller Würfel, der höher positioniert ist, wird anders wahrgenommen als ein dunklerer Würfel. Achte darauf, Lichtquellen so zu platzieren, dass sie den Schrumpf- und Vergrößerungseffekt visuell unterstützen, ohne dass sie den Eindruck der Position verfälschen.
- **Interaktive Elemente und physikalische Interaktionen:**
    - Bei der Positionierung der Objekte auf unterschiedlichen Höhen und Entfernungen sollten Interaktionen mit der Spielwelt berücksichtigt werden. Kollisionen oder Bewegungen der Würfel oder Kugeln können die Illusion stören. Stelle sicher, dass die Objekte keine unerwünschten physischen Reaktionen zeigen, die die Täuschung unterbrechen könnten.
- **Psychologische Wahrnehmung und Erwartungen:**

- Beachte, dass der Spieler mit der Zeit beginnen könnte, Muster in der Positionierung und der Größenverzerrung zu erkennen. Um dies zu vermeiden, variiere gelegentlich die Anordnung der Objekte und die Entfernungen, um das Gefühl der Täuschung aufrechtzuerhalten und Überraschungseffekte zu erzielen.

3.1.2 Dynamische Größenveränderung (2)

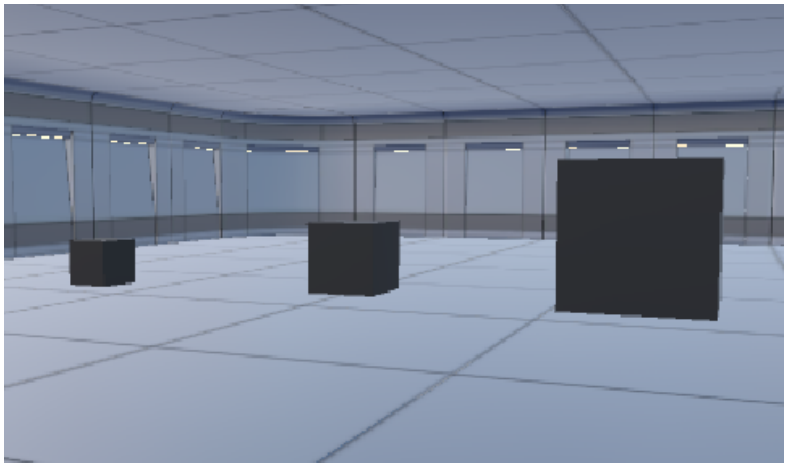

Abbildung 12: Dynamische Größenveränderung

Umsetzung:

Objekt

- Erstelle einen Würfel
- Füge das Skript DynamicSizeIllusion.cs dem Würfel als Komponente hinzu
  - Pfad: Assets → Scripts → SizeIllusion→ DynamicSizeIllusion
  - Wähle die Art der dynamischen Größenveränderung

Skript

DynamicSizeIllusion.cs

Pfad: Assets → Scripts → SizeIllusion → DynamicSizeIllusion.cs

```
    void Update()
    {
```

```
        float distance = Vector3.Distance(transform.position,
        playerCamera.position);

        float scaleFactor = baseScale;

        switch (behavior)
        {
            case SizeBehavior.ShrinkWhenNear:
                scaleFactor = (distance / referenceDistance) * baseScale;
                break;

            case SizeBehavior.EnlargeWhenNear:
                scaleFactor = (referenceDistance / distance) * baseScale;
                break;

            case SizeBehavior.Constant:
                scaleFactor = baseScale;
                break;
        }
        scaleFactor = Mathf.Clamp(scaleFactor, minScale, maxScale);
        transform.localScale = Vector3.one * scaleFactor;
    }
}
```

Wichtige Hinweise:

- **Kamera-Positionierung und Perspektive:**
  - Der Effekt der dynamischen Größenveränderung ist stark von der Position der Kamera und der Entfernung des Objekts abhängig. Achte darauf, dass die Kamera korrekt positioniert wird, um die Illusion zu verstärken. Wenn der Spieler sich zu schnell bewegt, könnte der Effekt der Größenveränderung abrupt oder unnatürlich wirken. Verhindere dies durch sanfte Übergänge in der Skalierung oder durch das Hinzufügen von Interpolationseffekten.
- **Wahl des Skalierungsverhaltens:**
  - Die Wahl des SizeBehavior (Shrinking, Enlarging oder Constant) beeinflusst die Wahrnehmung des Spielers. Bei einer ShrinkingWhenNear-Einstellung könnte die Wahrnehmung der Größe unklar werden, wenn das Objekt zu schnell zu klein wird. Andererseits könnte bei EnlargeWhenNear die Illusion von Überdimensionierung den Spieler verwirren. Teste verschiedene Varianten, um den gewünschten Effekt zu erzielen.

- **Skripterstellung und Performance:**
    - Die Berechnung der Skalierung in jedem Update()-Durchgang kann die Performance beeinträchtigen, wenn viele Objekte betroffen sind. Achte darauf, dass die dynamische Größenänderung nur auf Objekten angewendet wird, die es erfordern, und implementiere Culling-Mechanismen, um die Berechnungen auf nicht sichtbare Objekte zu reduzieren.
- **Skalierungs-Limits:**
    - Das Skalieren von Objekten hat Grenzen. Verwende Mathf.Clamp() für eine sinnvolle Begrenzung der maximalen und minimalen Größen, um sicherzustellen, dass Objekte nicht zu klein werden, um noch wahrnehmbar zu sein, oder so groß, dass sie durch die Spielwelt glitchen oder die Illusion zerstört wird. Experimentiere mit den Parametern minScale und maxScale, um die Grenzen für jedes Objekt individuell anzupassen.
- **Visuelle Artefakte und Übergänge:**
    - Wenn ein Objekt in der Größe stark verändert wird, können unerwünschte visuelle Artefakte auftreten, insbesondere bei komplexeren Modellen. Achte darauf, dass die Texturen und Meshes des Objekts auch bei großen Skalierungen noch gut sichtbar sind und keine Verzerrungen auftreten. Setze auch auf Interpolation (z.B. Lerp) für eine sanftere Übergangsanimation der Skalierung, um die visuelle Qualität zu verbessern.
- **Wahrnehmungspsychologie des Spielers:**
    - Dynamische Größenveränderung kann starke psychologische Effekte auf den Spieler haben, besonders in Kombination mit schnellen Bewegungen oder Interaktionen. Überlege, ob die Veränderung der Größe auf eine Art und Weise erfolgt, die den Spieler überrascht oder eine gewünschte Reaktion (z.B. Staunen, Verwirrung oder Bedrohung) hervorruft. Variiere gegebenenfalls den Effekt, um das Interesse des Spielers aufrechtzuerhalten.
- **Testen mit verschiedenen Entfernungen:**
    - Da der Abstand der Kamera zum Objekt die Wahrnehmung der Größenveränderung stark beeinflusst, teste die Implementierung aus verschiedenen Blickwinkeln und Entfernungen, um sicherzustellen, dass der Effekt korrekt wahrgenommen wird und nicht durch zu große oder zu kleine Skalierungen entstellt wird.
- **Interaktionen mit der Umgebung:**
    - Achte darauf, dass die dynamische Größenveränderung keine Konflikte mit der Umgebung oder anderen Gameplay-Mechaniken verursacht. Beispielsweise könnte ein Objekt, das vergrößert wird, plötzlich mit anderen Objekten oder Wänden kollidieren, was die Illusion zerstören könnte. Verwende Kollisionserkennungen und physikalische Interaktionen, die die Wahrnehmung der Größenveränderung unterstützen.
- **Bedeutung des Spielraums:**
    - Teste, ob der Raum ausreichend groß ist, damit sich die Objekte ohne Probleme vergrößern oder verkleinern können. Ein zu enger Raum könnte den visuellen Effekt behindern oder das Spielgefühl beeinträchtigen.

3.1.3 Licht und Farbe (1)

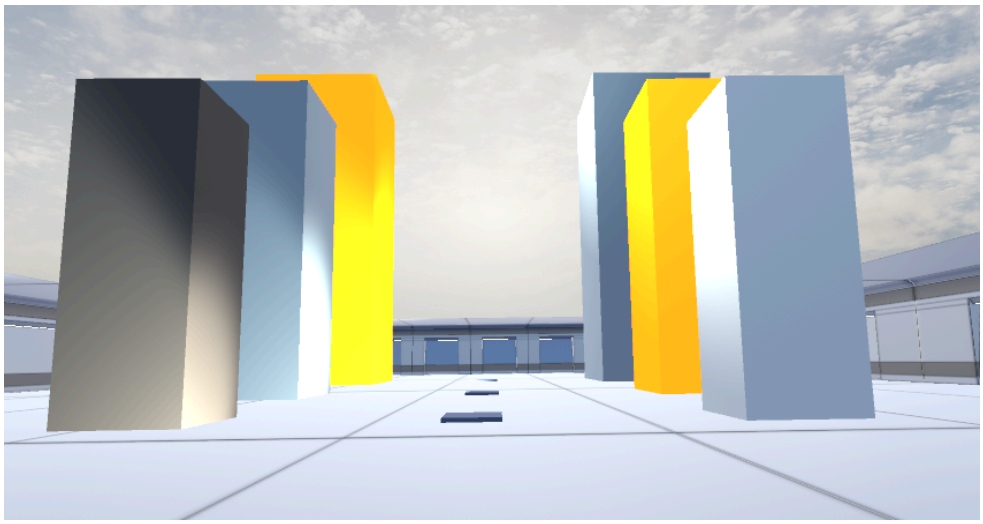

Abbildung 13: Einfluss von Licht und Farbe auf die Größe

Umsetzung:

Objekte

- Erstelle einen Würfel oder ein anderes Objekt deiner Wahl
  - **Objekt größer wirken lassen:**
    - Erstelle ein Spotlight und platziere es so, dass es das Objekt von unten bescheint
    - nutze ein Material mit warmer Farbe wie Gelb, Rot oder Orange
  - **Objekt kleiner wirken lassen:**
    - Erstelle ein Spotlight und platziere es so, dass es das Objekt von oben bescheint
    - nutze ein Material mit kühler Farbe wie Blau, Weiß oder Blaugrün

---

Wichtige Hinweise:

- **Wahrnehmung der Lichtquelle:**
  - Die Platzierung der Lichtquelle beeinflusst stark, wie ein Objekt wahrgenommen wird. Ein Licht von unten kann ein Objekt größer erscheinen lassen, indem es die unteren Flächen stärker betont, während ein Licht von oben die Schatten auf der Oberseite verstärkt und das Objekt kleiner erscheinen lässt. Experimentiere mit

der Intensität und dem Winkel der Lichtquelle, um den gewünschten Effekt zu erzielen.

- **Farben und ihre psychologischen Effekte:**
  - Farben haben eine starke psychologische Wirkung auf den Betrachter. Warme Farben (Rot, Gelb, Orange) neigen dazu, Objekte größer erscheinen zu lassen, da sie oft mit Nähe oder Wärme assoziiert werden. Kühler Farben (Blau, Weiß, Blaugrün) vermitteln häufig den Eindruck von Distanz und können die Illusion von Kleinheit verstärken. Achte darauf, dass die verwendeten Farben zum Gesamtkonzept und der Stimmung des Spiels passen.
- **Farbkontraste nutzen:**
  - Der Kontrast zwischen Licht und Schatten ist entscheidend, um die Illusion von Größenveränderungen zu verstärken. Wenn du ein Objekt größer erscheinen lassen möchtest, erhöhe den Kontrast der unteren Beleuchtung, sodass die Schatten klarer definiert sind. Bei kleineren Objekten kannst du einen weniger kontrastreichen, diffusen Lichtschein verwenden, um die Illusion zu unterstützen.
- **Materialwahl und Oberflächenreflexion:**
  - Materialien und ihre Reflexionseigenschaften beeinflussen, wie Licht auf ein Objekt trifft. Glänzende Oberflächen reflektieren mehr Licht und können die Größe eines Objekts verstärken, während matte Oberflächen Licht absorbieren und das Objekt kleiner erscheinen lassen. Wähle Materialien, die den gewünschten Effekt unterstützen. Achte darauf, dass die Materialien realistisch wirken, um die Illusion zu stärken.
- **Beleuchtungseffekte anpassen:**
  - Verwende nicht nur punktuelle Lichtquellen (wie Spotlights), sondern experimentiere auch mit Umgebungslicht und gerichteten Lichtern, um eine atmosphärische Wirkung zu erzielen. Ein globales Umgebungslicht kann die Gesamtwirkung der Beleuchtung ausbalancieren, sodass die starken Effekte der direkten Lichtquellen nicht zu überwältigend sind. So kannst du subtile und realistische Größenverzerrungen erzeugen.
- **Dynamische Lichtquellen:**
  - Bei dynamischen Szenen, in denen sich der Spieler oder das Objekt bewegt, sollte das Licht dynamisch angepasst werden, um konstante Wahrnehmungseffekte zu erzielen. Stelle sicher, dass die Lichtquellen in Echtzeit korrekt reagieren, wenn sich die Position des Objekts oder der Kamera ändert, um die Illusion der Größenveränderung aufrechtzuerhalten.
- **Licht und Schatten als zusätzliches Mittel der Illusion:**
  - Wenn ein Objekt dunklere Schatten auf der Unterseite hat (durch eine Lichtquelle von oben), kann es kleiner wirken. Umgekehrt kann ein Objekt mit helleren Schatten auf der Unterseite größer wirken. Experimentiere mit der Intensität und Positionierung der Schatten, um die Größenwahrnehmung zu beeinflussen und die Illusion zu verstärken.
- **Reflexionen und Transparenz:**
  - Objekte, die glänzende oder transparente Oberflächen haben, reagieren stärker auf Licht und Umgebung. Achte darauf, dass diese Objekte in der Nähe von

Lichtquellen korrekt reflektiert werden, um die Wahrnehmung der Größe zu beeinflussen. Verwende für solche Objekte gegebenenfalls Shader, die die Lichtverhältnisse dynamisch anpassen.

- **Kombination von Beleuchtung und Farbe:**
  - Die Kombination von Licht und Farbe kann auch einen dramatischen Einfluss auf die visuelle Illusion haben. Nutze beispielsweise ein warmes Licht in Kombination mit einer kühlen Farbe auf dem Material, um eine subtile visuelle Verzerrung zu erzeugen. So kannst du sowohl die Wahrnehmung der Größe als auch die der Tiefe im Raum verstärken.
- **Testen der visuellen Illusion:**
  - Teste die Wirkung der Beleuchtungseinstellungen aus verschiedenen Blickwinkeln und Entfernungen. Die Wahrnehmung von Größe kann sich je nach Perspektive des Spielers stark verändern. Überprüfe auch, wie die Beleuchtung unter verschiedenen Bedingungen (z.B. bei wechselndem Umgebungslicht oder bei Bewegung) wirkt, um sicherzustellen, dass die Illusion konsistent bleibt.

3.2 Allgemeine Tipps & Warnungen

Sanfte Übergänge:

- Bei der Implementierung von Größenillusionen ist es wichtig, dass die Übergänge zwischen den verschiedenen Wahrnehmungen möglichst sanft erfolgen. Harte oder abrupte Änderungen in der Größe können den Spieler verwirren oder die Illusion zerstören. Verwende interpolierte Skalierungen oder sanfte Animationen, um die Wahrnehmung der Größenveränderung angenehm und glaubwürdig zu gestalten.

Kontextabhängigkeit:

- Achte darauf, dass Größenillusionen im Kontext der Spielwelt sinnvoll sind. Zu häufige oder unpassende Größenänderungen können die Immersion des Spiels verringern und den Spieler aus der Erfahrung reißen. Setze Größenillusionen gezielt ein, wenn sie zur Stimmung, den Rätseln oder der Story des Spiels beitragen.

Testen unter verschiedenen Bedingungen:

- Um sicherzustellen, dass Größenillusionen im gesamten Spiel wirken, teste sie unter verschiedenen Blickwinkeln, Entfernungen und in verschiedenen Umgebungen. Der Effekt kann je nach Perspektive des Spielers variieren. Dies gilt besonders für dynamische Szenen, in denen sich der Spieler oder die Objekte bewegen.

Berücksichtigung der Spielerinteraktion:

- Größenillusionen sollten nicht die Steuerung oder das Gameplay unnötig komplizieren. Wenn ein Objekt durch eine Größenillusion manipuliert wird, achte darauf, dass der Spieler die Interaktion damit immer noch intuitiv und ohne Frustration durchführen kann. Unvorhersehbare Größenänderungen können dazu führen, dass Spieler die Steuerung als unangemessen empfinden.

Vermeidung von Übertreibungen:

- Achte darauf, dass die Größenillusionen nicht zu extrem ausfallen. Eine übermäßige Verzerrung kann den Effekt entkräften und zu Desorientierung führen. Mäßigung ist entscheidend, um die Illusion glaubwürdig und nicht lächerlich wirken zu lassen.

Optimierung der Performance:

- Größere Objekte und viele gleichzeitig aktive Illusionen können die Performance des Spiels beeinträchtigen. Achte darauf, dass du bei der Verwendung von Größenillusionen die Renderleistung im Auge behältst und gegebenenfalls Optimierungen vornimmst, um die Framerate konstant hoch zu halten.

Visuelle Konsistenz:

- Um die Wirkung von Größenillusionen zu maximieren, stelle sicher, dass der Rest der Szene visuell konsistent bleibt. Unstimmigkeiten in der Beleuchtung, den Materialien oder der Textur können die Wahrnehmung der Größenveränderung stören.

Vermeidung von Verwirrung:

- Achte darauf, dass die Illusionen nicht so stark ausgeprägt sind, dass sie den Spieler verwirren oder frustrieren. Die Illusion sollte eine ergänzende Wirkung auf das Gameplay haben und keine barrierebehaftete Herausforderung darstellen.

3.3 Verwendungszweck

Erkundung und Rätsel:

- Größenillusionen eignen sich hervorragend für Spiele, die auf Erkundung und Problemlösung basieren. In Spielen wie Superliminal und Portal können Größenillusionen genutzt werden, um den Spieler zu verwirren und gleichzeitig kreative Lösungen für Rätsel zu entwickeln. Objekte können durch perspektivische Verzerrungen manipuliert werden, um neue Wege zu öffnen oder Hindernisse zu überwinden.

Immersion und Atmosphäre:

- Größenillusionen sind ein kraftvolles Werkzeug, um die Immersion in einer Spielwelt zu verstärken. Sie können genutzt werden, um eine surreale oder unheimliche Atmosphäre zu erzeugen, wie es oft in psychologischen Horrorspielen der Fall ist. Ein Beispiel ist die verzerrte Wahrnehmung von Dimensionen in Spielen wie Antichamber oder Layers of Fear, bei denen sich die Spielwelt ständig verändert und die Wahrnehmung des Spielers auf die Probe gestellt wird.

Visuelles Storytelling:

- Größenillusionen können auch eine bedeutende Rolle im Storytelling spielen. Sie können verwendet werden, um Themen wie Macht, Größenverhältnisse oder ungleiche Beziehungen darzustellen. So können bestimmte Objekte oder Charaktere übergroß erscheinen, um deren Bedeutung oder Bedrohung zu betonen. Andererseits kann eine kleine Darstellung von Charakteren oder Objekten die Verletzlichkeit oder Unwichtigkeit verdeutlichen.

Psychologische Effekte:

- Größenillusionen können gezielt eingesetzt werden, um psychologische Reaktionen wie Angst, Verwirrung oder Staunen hervorzurufen. In Horrorspielen wie Amnesia oder

Inside können verzerrte Größenverhältnisse den Spieler in einen Zustand der Unsicherheit versetzen und das Gefühl von Bedrohung oder Unwohlsein verstärken.

Spielmechaniken und Herausforderungen:

- Größenillusionen können als innovative Spielmechanik eingesetzt werden, um neue Herausforderungen zu schaffen. In Spielen, die auf Geschicklichkeit und Timing angewiesen sind, kann die Wahrnehmung von Größe genutzt werden, um den Spieler zu zwingen, seine Perspektive oder seine Taktiken anzupassen. Die Illusion von sich verändernden Größen kann auch dazu beitragen, die Schwierigkeit eines Spiels dynamisch zu verändern.

Künstlerische und ästhetische Gestaltung:

- Größenillusionen bieten auch kreative Freiheiten für Entwickler, um künstlerische und ästhetische Akzente zu setzen. Durch die Kombination von Licht, Farbe und Perspektive können atemberaubende visuelle Effekte erzielt werden, die das Spiel von anderen abheben. Diese Illusionen können besonders in Szenen verwendet werden, die den Spieler visuell beeindrucken oder emotional ansprechen sollen.

Versteckte Elemente und Geheimnisse:

- Größenillusionen können dazu verwendet werden, versteckte Räume oder Geheimnisse zu präsentieren, die nur aus bestimmten Blickwinkeln oder Entfernungen sichtbar sind. Dies fördert die Erkundung und belohnt den Spieler für die Entdeckung von Details in der Spielwelt. Diese Technik kann in Spielen wie Shadow of the Colossus oder Viewfinder eingesetzt werden, um das Gefühl einer lebendigen und geheimnisvollen Welt zu verstärken.

4 Psychologische Illusionen

4.1 Technische Implementierung

4.1.1 Tunnelblick (2)

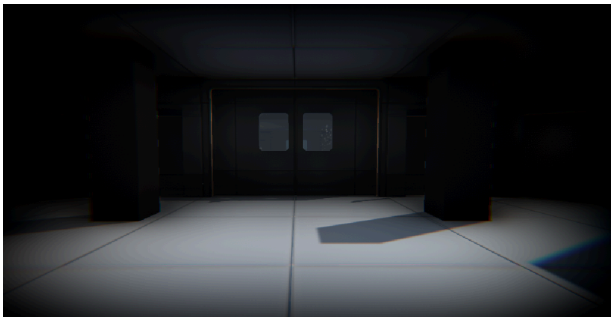

Abbildung 14: Tunnelblick

Umsetzung:

Objekt:

- Erst ein Empty Game Objekt erstellen, Box Collider und Post Process Volume hinzufügen.

Post Process:

- **Vignette:**
    - Setze die Intensität auf ca. 0.6
        - Dadurch soll die Sicht beschränkt werden
    - Setze Smoothness auf 1
        - Ermöglicht einen weichen Übergang
    - Setze den Roundness Wert auf ca. 0.8
        - Wirkung eines Tunnelblicks erzeugen
- **Chromatic Aberration**
    - Erhöhe die Intensität (z.B. auf 0.8)
        - Erzeugt rote/blaue und grüne/violette Farbsäume
- **Ambient Occlusion**
    - Erhöhe die Intensität auf ca. 1.5
        - Dies dunkelt Flächen ab, die nahe beieinander liegen
- **Layer-Zuweisung:**
    - Erstelle einen neuen Layer (z. B. „PostProcessing“) und weise es dem Volume-Objekt zu.
    - Stelle sicher, dass deine Kamera den richtigen Layer im Post Process Layer-Component ausgewählt hat.

Wichtige Hinweise:

- **Dosierung der Vignette:**
    - Achte darauf, wie stark die Vignette die Sicht des Spielers einschränkt. Zu hohe Intensität kann den Spieler desorientieren und das Gameplay negativ beeinflussen. Es empfiehlt sich, mit unterschiedlichen Werten zu experimentieren, um den richtigen Grad der Einschränkung zu finden, ohne dass der Tunnelblick zu belastend wird.
- **Rechteckiger Tunnelblick:**
    - Wenn du einen stärker definierten Tunnelblick erzeugen möchtest, kannst du zusätzlich zu den Vignetten-Einstellungen mit einer Form-Änderung experimentieren, indem du z. B. die Vignette rund machst oder asymmetrische Verzerrungen hinzufügst.
- **Performance:**
    - Post-Processing-Effekte wie Vignette, Chromatic Aberration und Ambient Occlusion können die Performance beeinträchtigen, insbesondere wenn sie in Echtzeit auf allen Szenen angewendet werden. Stelle sicher, dass die Performance des Spiels nicht unter diesen Effekten leidet, indem du

Optimierungen vornimmst oder diese Effekte nur in bestimmten Szenen oder bei Bedarf einsetzt.

- **Rückmeldung des Spielers:**
  - Teste regelmäßig, wie sich der Tunnelblick auf das Spielerlebnis auswirkt. Zu starke visuelle Verzerrungen können die Spieler verwirren oder ihnen das Gefühl geben, die Kontrolle zu verlieren, was je nach Kontext erwünscht oder ungewünscht sein kann. Daher sollte der Effekt nur in Szenen verwendet werden, in denen er das gewünschte psychologische Erlebnis verstärkt.
- **Einstellbare Intensität:**
  - Biete dem Spieler die Möglichkeit, die Intensität des Tunnelblicks in den Optionen des Spiels anzupassen, falls der Effekt zu störend wird. Dies sorgt dafür, dass Spieler die Kontrolle über ihre Wahrnehmung behalten und nicht durch störende visuelle Effekte das Gefühl haben, das Spiel zu verlassen.

4.1.2 Wärmeempfinden (3)

4.1.2.1 Profil

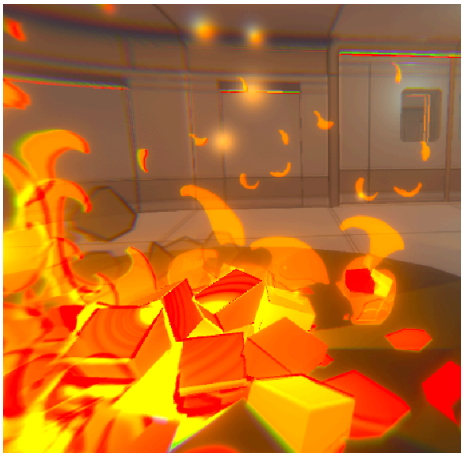

Abbildung 15: Wärmeempfindung

Umsetzung:

Profil:

- **Profil erstellen:**
  - Rechtsklick im Projektfenster → Create → Post Processing Profile
  - Benenne das Profil z. B. HeatProfile“.
- **Profil öffnen:**

    - Wähle das neu erstellte Profil aus, damit du im Inspector die enthaltenen Overrides bearbeiten kannst.
- **Color Grading Override hinzufügen:**
    - **Temperature:**
        - Aktiviere das Override (Checkbox) und setze z. B. einen Wert von +20 bis +30. Dies sorgt für einen wärmeren Farbton.
    - **Tint:**
        - Optional: Leichte Anpassung (z. B. +5 bis +10) kann den Rot- bzw. Gelbstich verstärken.
    - **Saturation:**
        - Erhöhe die Sättigung (z. B. +10 bis +20), um die Farben lebendiger und intensiver wirken zu lassen.
    - **Contrast:**
        - Ein leichter Kontrastanstieg (z. B. +5 bis +10) kann zusätzlich das dynamische Aussehen unterstützen.
- **Bloom Override hinzufügen:**
    - **Intensity:**
        - Setze den Wert z. B. zwischen 1.0 und 2.0. Je näher der Spieler an das Feuer kommt, desto höher sollte dieser Wert werden.
    - **Threshold:**
        - Ein Wert um 1.0 sorgt dafür, dass nur sehr helle Bereiche den Bloom-Effekt auslösen.
    - **Soft Knee:**
        - Etwa 0.5 für einen sanften Übergang zwischen hellen und weniger hellen Bereichen.
- **Chromatic Aberration Override hinzufügen:**
    - **Intensity:**
        - Beginne mit einem Wert von 0.2 bis 0.3. Ein leichter Verzerrungseffekt kann das Flimmern der Hitze andeuten.

Post Process Volume:

- **Volume-Objekt anlegen:**
    - Erstelle oder wähle das GameObject (z. B. „RoomVolume“), das deinen Hitze-Bereich repräsentiert.
    - Füge diesem GameObject die Post Process Volume-Komponente hinzu.
- **Volume-Einstellungen:**
    - **Profile:** Weise hier dein „HeatEffectProfile“ zu.
    - **Is Global:** Deaktiviere diese Option, wenn der Effekt nur lokal wirken soll.
    - **Weight:** Setze auf **1**, damit der Effekt vollständig angewendet wird.
    - **Blend Distance:** Wähle einen sinnvollen Wert, damit der Übergang in den Effektbereich weich erfolgt.
- **Layer-Zuweisung:**
    - Erstelle einen neuen Layer (z. B. „PostProcessing“) und weise es dem Volume-Objekt zu.

- Stelle sicher, dass deine Kamera den richtigen Layer im Post Process Layer-Component ausgewählt hat.

4.1.3.2 Post Processing Effekt

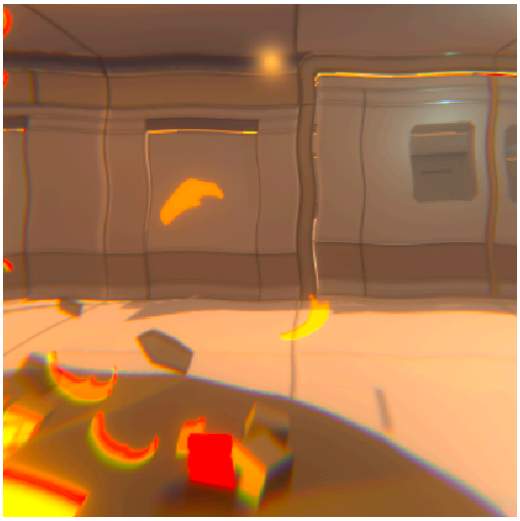

Abbildung 16: Hitzewellen

Hitzeverzerrung als Post Process Effekt:

- lege ein neues Skript für den Renderer und den Post Processing Effekt an
- Erstelle einen neuen Shader

Skript:

HeatDistortion.cs:

Pfad: Assets → Scripts→ PsychologicalIllusion → HeatDistortion.cs

```
using UnityEngine;
using UnityEngine.Rendering.PostProcessing;

[System.Serializable]
[PostProcess(typeof(HeatDistortionRenderer), PostProcessEvent.BeforeStack,
"Hidden/Custom/HeatDistortion")]
public sealed class HeatDistortion : PostProcessEffectSettings
{
    [Range(0f, 0.1f), Tooltip("Amplitude der Verzerrung.")]
    public FloatParameter amplitude = new FloatParameter { value = 0.0f };

    [Range(0f, 100f), Tooltip("Frequenz der Verzerrung.")]
```

```
    public FloatParameter frequency = new FloatParameter { value = 0f };

    [Range(0f, 10f), Tooltip("Geschwindigkeit der Verzerrung.")]
    public FloatParameter speed = new FloatParameter { value = 0f };
}
```

HeatDistortionRenderer.cs:

Pfad: Assets → Scripts→ PsychologicalIllusion → HeatDistortionRenderer.cs

```
using UnityEngine;
using UnityEngine.Rendering.PostProcessing;

public sealed class HeatDistortionRenderer :
PostProcessEffectRenderer<HeatDistortion>
{
    Shader shader;

    public override void Init()
    {
        shader = Shader.Find("Hidden/Custom/HeatDistortion");
    }

    public override void Render(PostProcessRenderContext context)
    {
        var sheet = context.propertySheets.Get(shader);
        sheet.properties.SetFloat("_Amplitude", settings.amplitude);
        sheet.properties.SetFloat("_Frequency", settings.frequency);
        sheet.properties.SetFloat("_Speed", settings.speed);
        context.command.BlitFullscreenTriangle(context.source,
        context.destination, sheet, 0);
    }
}
```

HeatDistortion.shader:

Pfad: Assets → Shaders → HeatDistortion.shader

```
Shader "Hidden/Custom/HeatDistortion" {
    Properties {
        _Amplitude ("Amplitude", Float) = 0
        _Frequency ("Frequency", Float) = 0
```

```
        _Speed ("Speed", Float) = 0
    }
    SubShader {
        Tags { "RenderType"="Opaque" }
        Pass {
            ZTest Always Cull Off ZWrite Off
            HLSLPROGRAM

            #pragma vertex vert
            #pragma fragment frag
            #include "UnityCG.cginc"

            sampler2D _MainTex;
            float _Amplitude;
            float _Frequency;
            float _Speed;

            struct v2f {
                float4 pos : SV_POSITION;
                float2 uv  : TEXCOORD0;
            };

            v2f vert(uint vertexID : SV_VertexID) {
                v2f o;
                float2 pos[3];
                pos[0] = float2(-1.0, -1.0);
                pos[1] = float2( 3.0, -1.0);
                pos[2] = float2(-1.0,  3.0);
                o.pos = float4(pos[vertexID], 0.0, 1.0);
                o.uv = (o.pos.xy + 1.0) * 0.5;
                return o;
            }

            float4 frag(v2f i) : SV_Target {
                float2 offset;
                offset.x = sin(i.uv.y * _Frequency + _Time.y * _Speed) *
                _Amplitude;
                offset.y = cos(i.uv.x * _Frequency + _Time.y * _Speed) *
                _Amplitude;
                float2 uvDistorted = i.uv + offset;

                float2 uvFixed = uvDistorted;
                #if defined(UNITY_UV_STARTS_AT_TOP)
```

```
                    uvFixed.y = 1.0 - uvFixed.y;
                #endif

                return tex2D(_MainTex, uvFixed);
            }
            ENDHLSL
        }
    }
}
```

- Der HeatDistortion Effekt kann nun im als Overright dem Profil hinzugefügt werden
    - Pfad: Hidden → Custom → HeatDistortion

Wichtige Hinweise:

- **Wärmeempfindung anpassen:**
    - Achte darauf, dass der Effekt nicht zu aufdringlich wird. Eine zu hohe Intensität kann den Spieler desorientieren oder das Gameplay unnötig erschweren. Teste unterschiedliche Werte, um die Balance zwischen realistischer Wirkung und Spielbarkeit zu finden.
- **Performance:**
    - Die Nutzung von Post-Processing-Effekten wie Bloom, Chromatic Aberration und Verzerrung kann die Performance, insbesondere bei mobilen Geräten, negativ beeinflussen. Überprüfe regelmäßig die Framerate, um sicherzustellen, dass der Effekt nicht zu stark belastet.
- **Anpassungen für den Kontext:**
    - Der Wärmeeffekt sollte in Bereichen verwendet werden, die eine intensive, heißere Umgebung simulieren, wie z.B. in der Nähe von Feuer oder Lava. In kühleren Szenen könnte der Effekt irritierend wirken.
- **Einstellung der Amplitude:**
    - Übermäßige Amplitude in der Verzerrung kann das Spielerlebnis unangenehm machen. Eine leichte Verzerrung wirkt subtil und ist oft effektiver als eine starke Verzerrung.

Genutzte Tutorials und Assets:

- **Tutorial**:
    - Benjamin Swee - Custom Unity Shaders. (23.04.2022). Custom Unity Distortion Shader Using Post Processing [Video]. YouTube. Verfügbar unter: https://youtu.be/YSAqivxqC3w?si=JLe4Emr6uiVZtED2
- **Assets:**

- WALLCOEUR. (letzte Veröffentlichung: 20.12.2024). FX URP - Fire Package [Asset]. Unity Asset Store. Verfügbar unter: https://assetstore.unity.com/packages/vfx/particles/fire-explosions/vfx-urp-fire-package-305098

Empfehlung:

- Mech Mind Games. (21.04.2021). HOW TO MAKE DISTORTION/HEAT EFFECT FOR 2D & 3D - SIMPLE SHADER GRAPHS - Unity 2019 LTS [Video]. YouTube. Verfügbar unter: https://youtu.be/rSiYL04H1jE?si=weaL2QSog8ioyqc8
- Gabriel Aguiar Prod.. (09.02.2021). Unity VFX Graph - Heat Distortion Effect Tutorial [Video]. YouTube. Verfügbar unter: https://youtu.be/CXCyVDEplyM?si=aXpGsZ4pS9rgaOLU
- PolyToots. (19.08.2020). Unity/Amplify: Heat Haze Distortion, Boiling Bubbles, Fire/Smoke and God Rays! [Video]. YouTube. Verfügbar unter: https://youtu.be/jFkhFuc0H4k?si=E28PBD0T4WEKhhmq

---

4.1.3 Kälteempfinden (2)

4.1.3.1 Profil

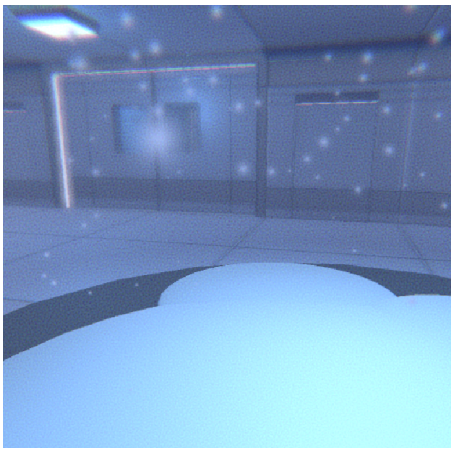

Abbildung 17: Kälteempfindung

Umsetzung:

Profil:

- **Profil erstellen:**
  - Rechtsklick im Projektfenster → Create → Post Processing Profile
- **Profil öffnen:**

    - Wähle das neu erstellte Profil aus, damit du im Inspector die enthaltenen Overrides bearbeiten kannst.
- **Color Grading Override hinzufügen:**
    - **Temperature:**
        - Aktiviere das Override und setze einen negativen Wert (z. B. von 0 bis -20).
        - Je kälter die Umgebung wirken soll, desto mehr in den negativen Bereich gehen.
    - **Tint:**
        - Eine leichte Verschiebung in Richtung Blau oder Grün (z. B. -5 bis -10) verstärkt den Kälte-Eindruck.
    - **Saturation:**
        - Du kannst die Sättigung leicht reduzieren, damit die Farben matter und kälter wirken.
    - **Contrast:**
        - Ein etwas höherer Kontrast (z. B. +5) kann helfen, die Effekte zu betonen.
- **Vignette Override hinzufügen:**
    - Erhöhe die Intensität (z. B. 0.3 bis 0.5), um den Blick des Spielers stärker auf den zentralen Bereich zu lenken und das Gefühl von Isolation zu verstärken.
- **Film Grain Override hinzufügen:**
    - Erhöhe die Intensität (z. B. 0.3 bis 0.6), um einen körnigen, frostigen Look zu simulieren, der oft mit Kälte assoziiert wird.
- **Chromatic Aberration Override hinzufügen:**
    - Setze einen dezenten Wert (z. B. 0.1 bis 0.2), um eine leichte, kalte Verzerrung zu erzeugen. Bei Kälte sollte dieser Effekt in der Regel weniger ausgeprägt sein als bei Hitze.

Post Process Volume:

- **Volume-Objekt anlegen:**
    - Erstelle ein GameObject (z. B. „RoomColdVolume“) in deinem Kältebereich.
    - Füge diesem die Post Process Volume-Komponente hinzu.
- **Volume-Einstellungen:**
    - **Profile:** Weise dem Volume das „ColdEffectProfile“ zu.
    - **Is Global:** Deaktiviere diese Option, damit der Effekt nur lokal wirkt.
    - **Weight:** Stelle den Wert auf 1.
    - **Blend Distance:** Wähle einen sinnvollen Wert (z. B. 5–10 Einheiten), um einen weichen Übergang zum Normalzustand zu ermöglichen.
- **Layer-Zuweisung:**
    - Weise diesem Volume ein eigenes Layer (z. B. „PostProcessing“) zu, und stelle sicher, dass die Kamera den richtigen Layer im Post Process Layer-Component ausgewählt hat.

Schnee Partikel System:

Objekt:

- **Erstelle ein Empty Game Objekt**
    - Füge die Particle System Komponente hinzu

Particle System:

- Setze Start Speed auf 0
    - Dies ändert die Richtung in der die Partikel sich bewegen
- Setze Start Size auf ‘Random Between Two Constants’
    - wähle Werte zwischen 0,01 und 0,5
    - Hiermit werden die Partikel zufällig groß generiert
- **Emission:**
    - Setze Rate over Time auf 100
        - dies erhöht die Anzahl an Partikeln
- **Shape:**
    - Ändere unter Shape → Shape die Form zu Box
    - Passe die Größe der Fläche über Scale an (z.B. X: 10, Y:1, Z:10)
- **Velocity over Lifetime:**
    - Setze die Y-Achse unter ‘Random Between Two Constants’ auf einen negativen Wert (z.B. -1 und -2)
        - dies bewegt die Partikel in unterschiedlichen Geschwindigkeiten nach unten
- **Size over Lifetime:**
    - Passe die Kurve so an, dass die Partikel schnell an Größe gewinnen und danach schrumpfen
        - Dies ermöglicht einen weichen Übergang

Abbildung 18: Size over Lifetime Kurve

- **Noise:**
    - Verringere die Stärke (z.B. auf 0.2)

Wichtige Hinweise:

- **Farbkorrekturen fein abstimmen**:
    - Achte darauf, dass die Farbton- und Sättigungswerte für das Color Grading nicht zu extrem gesetzt sind, um unrealistische oder störende visuelle Eindrücke zu vermeiden. Die Kälte sollte subtil und dennoch spürbar sein, um das gewünschte Gefühl zu erzeugen.
- **Vignette und Film Grain mit Bedacht einsetzen**:
    - Obwohl Vignette und Film Grain dabei helfen können, das Kälteempfinden zu verstärken, können sie auch die Klarheit und das visuelle Erlebnis beeinträchtigen. Stelle sicher, dass die Intensität der Vignette und des Film Grains nicht zu hoch ist, besonders in Szenen, in denen Details wichtig sind.
- **Partikel-System optimieren**:
    - Bei der Verwendung des Schnee-Partikel-Systems ist es wichtig, darauf zu achten, wie viele Partikel gleichzeitig aktiv sind, um die Leistung nicht zu beeinträchtigen. Teste, ob die Performance bei verschiedenen Systemen (z. B. Low-End-PCs oder Mobilgeräten) stabil bleibt.
- **Schnee-Effekte als dynamisches Element**:
    - Der Schnee-Effekt kann die Wahrnehmung der Kälte verstärken, besonders wenn er in den Spielverlauf integriert wird (z. B. beim Betreten eines kalten Gebiets oder bei sich verändernden Wetterbedingungen). Berücksichtige daher, wie Partikel die Immersion unterstützen und interaktive Elemente wie Wind oder plötzliche Schneefälle dynamisch umsetzen.
- **Shader-Anpassungen für den Schnee**:
    - Überlege, ob der Schnee in der Umgebung auch Texturänderungen benötigt. Ein passender Shader für den Schnee kann helfen, die Reflexionen und das Gesamtbild zu verbessern.
- **Kompatibilität mit verschiedenen Beleuchtungsszenarien**:
    - Das Kälteempfinden kann stark von der Beleuchtung abhängen. Stelle sicher, dass der Kälte-Effekt in verschiedenen Beleuchtungsszenarien funktioniert, insbesondere wenn du mit tag- und nachtzeitabhängigen Lichtverhältnissen arbeitest. Teste, ob der negative Farbton und der Vignetten-Effekt auch unter unterschiedlichen Lichtquellen gut zur Geltung kommen.
- **Leistungsoptimierung**:
    - Denke daran, dass Post-Processing-Effekte und Partikelsysteme die Leistung auf älteren oder weniger leistungsfähigen Geräten negativ beeinflussen können. Nutze in solchen Fällen die „Quality Settings“ von Unity und teste die Performance gründlich, um sicherzustellen, dass dein Spiel auf verschiedenen Geräten flüssig läuft.
- **Erweiterbarkeit und Anpassbarkeit**:
    - Achte darauf, dass die Kälteempfindung auch erweiterbar bleibt, falls du weitere Effekte wie Wind, Eisschollen oder gefrorene Oberflächen hinzufügen möchtest. Plane die Shader und Partikelsysteme so, dass sie bei Bedarf flexibel angepasst oder erweitert werden können.
- **Anpassung der Blend-Distanz**:

- Die Blend-Distanz im Post-Processing-Volume bestimmt, wie abrupt der Kälte-Effekt einsetzt. Stelle sicher, dass diese Entfernung so gewählt wird, dass der Übergang zwischen normalen und kalten Bereichen nicht zu abrupt wirkt, sondern ein natürlicher Effekt entsteht.

Genutzte Tutorials und Assets:

- **Tutorial:**
  - Mirza. (08.01.2022). Snow Particle System Tutorial for Unity in just 2 Minutes [Video]. YouTube. Verfügbar unter: https://youtu.be/T_8cPlpTr90?si=P-nddWs7xdrL9hfl

---

4.1.4 Schmerzempfinden (2)

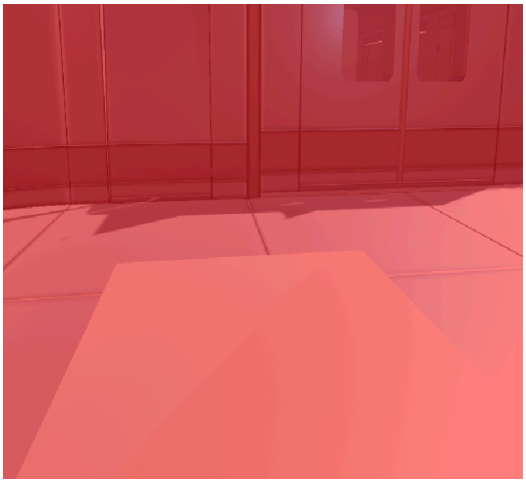

Abbildung 19: Schmerzempfinden Overlay

Umsetzung:
Overlay:

- Erstelle ein Canvas
  - setze den Render Mode auf Screen Space - Overlay
- Erstelle ein Image im Canvas und skaliere die Größe, sodass es den Canvas füllt.
  - Setze die Farbe auf rot (z.B. RGB: 255, 13, 13)

Objekt:

- Erstelle ein Cube oder ein anderes Objekt deiner Wahl, dass den Schmerz Effekt auslösen soll

Zittern:

- Füge das PainEffect.cs Skript an

Skript:

PainEffect.cs

Pfad: Assets → Scripts → PsychologicalIllusion → PainEffect.cs

```
    private void OnTriggerEnter(Collider other)
    {
        if (other.CompareTag("Player"))
        {
            StartCoroutine(ActivatePainEffect());
            StartCoroutine(ShakeCamera());
        }
    }

    private IEnumerator ActivatePainEffect()
    {
        redOverlay.gameObject.SetActive(true);
        Color color = redOverlay.color;
        color.a = 0.5f;
        redOverlay.color = color;

        yield return new WaitForSeconds(effectDuration);
        StartCoroutine(FadeOutOverlay(0.5f));
    }


    private IEnumerator FadeOutOverlay(float fadeDuration)
    {
        redOverlay.gameObject.SetActive(true);
        Color color = redOverlay.color;
        color.a = 1f;
        redOverlay.color = color;

        float elapsed = 0f;
        while (elapsed < fadeDuration)
        {
            elapsed += Time.deltaTime;
            float newAlpha = Mathf.Lerp(0.5f, 0f, elapsed / fadeDuration);
            color.a = newAlpha;
            redOverlay.color = color;
            yield return null;
```

```
        }
        color.a = 0f;
        redOverlay.color = color;
        redOverlay.gameObject.SetActive(false);
    }

    private IEnumerator ShakeCamera()
    {
        if (isShaking) yield break;
        isShaking = true;
        Vector3 originalPos = Camera.main.transform.localPosition;
        float elapsed = 0f;
        while (elapsed < shakeDuration)
        {
            float offsetX = Random.Range(-shakeIntensity, shakeIntensity);
            float offsetY = Random.Range(-shakeIntensity, shakeIntensity);
            Camera.main.transform.localPosition = originalPos + new
            Vector3(offsetX, offsetY, 0);
            elapsed += Time.deltaTime;
            yield return null;
        }
        Camera.main.transform.localPosition = originalPos;
        isShaking = false;
    }
}
```

Trigger-Ereignis (OnTriggerEnter):

- Prüft, ob das kollidierende Objekt den Tag "Player" hat.
- Startet bei einer Kollision zwei Coroutinen:
    - *ActivatePainEffect()* für den visuellen Schmerz-Effekt.
    - *ShakeCamera()* für die Kamerawackel-Animation.

Aktivieren des Schmerz-Effekts (ActivatePainEffect):

- Aktiviert das rote Overlay-Objekt und setzt dessen Transparenz (Alpha) auf 0.5, um einen halbtransparenten Effekt zu erzielen.
- Wartet für die Dauer *effectDuration*.
- Startet anschließend die Coroutine *FadeOutOverlay(0.5f)*, um den Overlay langsam auszublenden.

Ausblenden des Overlays (FadeOutOverlay):

- Aktiviert erneut das rote Overlay und setzt dessen Alpha-Wert zunächst auf 1.

- Über einen Zeitraum von *fadeDuration* Sekunden wird in einer Schleife der Alpha-Wert mittels *Mathf.Lerp* schrittweise von 0.5 auf 0 interpoliert, um einen sanften Fade-Out-Effekt zu erzeugen.
- Nach der Schleife wird der Alpha-Wert final auf 0 gesetzt und das Overlay deaktiviert.

Kamerawackeln (ShakeCamera):

- Verhindert, dass mehrere gleichzeitige Wackel-Effekte gestartet werden, indem geprüft wird, ob bereits gewackelt wird (*isShaking*).
- Speichert die ursprüngliche Position der Kamera.
- Während der festgelegten *shakeDuration* werden zufällige Offsets in X- und Y-Richtung erzeugt und zur ursprünglichen Position addiert, um den Wackeleffekt zu simulieren.
- Nach Ablauf der Zeit wird die Kamera wieder an ihre ursprüngliche Position zurückgesetzt und der Wackeleffekt beendet (*isShaking* wird auf false gesetzt).

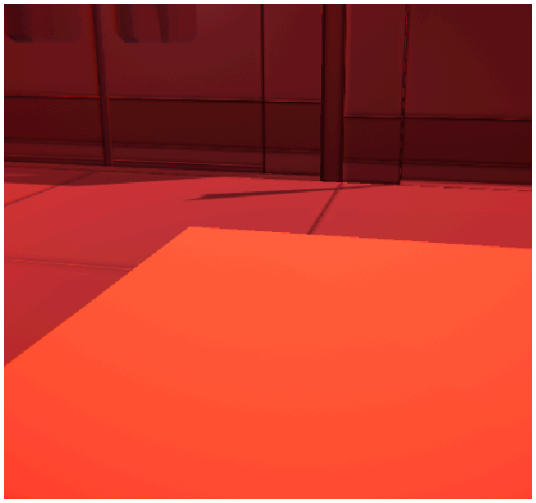

Abbildung 20: Schmerzempfinden Post Processing

Umsetzung:

Schmerz Profil:

- **Profil erstellen:**
  - Rechtsklick im Projektfenster → Create → Post Processing Profile
- **Profil öffnen:**
  - Wähle das neu erstellte Profil aus, damit du im Inspector die enthaltenen Overrides bearbeiten kannst.
- **Vignette Override hinzufügen:**
  - **Color:**

        - Aktiviere das Override und setze eine dunkelrote Farbe
    - **Intensity:**
        - Passe die Intensität an (z.B. +0.5)
- **Color Grading Override hinzufügen:**
    - **Color Filter:**
        - Setze die Farbe auf rot
            - passe die Farbe nach bedarf an, je nachdem wie hoch der Schmerzgrad ist kann die Farbe intensiver gewählt werden
    - **Saturation:**
        - die Sättigung leicht erhöhen, damit die Farben glänzender und schärfer wirken.
- **Chromatic Aberration Override hinzufügen:**
    - Setze einen hohen Wert (z. B. 08 bis 1), um eine starke, schmerzhafte Verzerrung zu erzeugen.

Post Process Volume

- **Volume-Objekt anlegen:**
    - Erstelle ein GameObject (z. B. „PainVolume“) als Kind des Objektes, das den Schmerz-Effekt auslösen soll .
    - Füge diesem die Post Process Volume-Komponente hinzu.
- **Volume-Einstellungen:**
    - **Profile:** Weise dem Volume das „PainEffectProfile“ zu.
    - **Is Global:** Deaktiviere diese Option, damit der Effekt nur lokal wirkt.
    - **Weight:** Stelle den Wert auf 1.
    - **Blend Distance:** Wähle einen sinnvollen Wert (z. B. 5–10 Einheiten), um einen weichen Übergang zum Normalzustand zu ermöglichen.
- **Layer-Zuweisung:**
    - Weise diesem Volume ein eigenes Layer (z. B. „PostProcessing“) zu, und stelle sicher, dass die Kamera den richtigen Layer im Post Process Layer-Component ausgewählt hat.

Wichtige Hinweise:

- **Farbintensität für den Schmerz-Effekt anpassen**:
    - Die rote Farbe für das Overlay und das Color Grading sollte je nach Schmerzintensität angepasst werden. Ein stärkerer Schmerz kann durch eine intensivere Rottönung und höhere Sättigung erreicht werden, während ein milder Schmerz durch eine sanftere Farbabstufung dargestellt werden kann.
- **Overlay-Transparenz dynamisch anpassen**:
    - Achte darauf, dass die Transparenz des roten Overlays nicht zu schnell oder zu langsam ausgeblendet wird. Der Wert für effectDuration sollte so gewählt

werden, dass der Übergang zu realistisch wirkt. Teste den Effekt in verschiedenen Szenen und unter verschiedenen Lichtverhältnissen, um sicherzustellen, dass der Effekt nicht zu aufdringlich ist.

- **Kamerawackeln realistischer gestalten**:
  - Der Kamerawackel-Effekt kann die Spielerfahrung intensivieren, sollte jedoch nicht zu störend wirken. Stelle sicher, dass die Wackel-Intensität (shakeIntensity) und -Dauer (shakeDuration) nicht zu extrem sind, damit der Effekt nicht die Spielfokus oder Orientierung des Spielers beeinträchtigt.
- **Performance im Blick behalten**:
  - Das Hinzufügen von Post-Processing-Effekten und Kamerawackeln kann die Performance auf weniger leistungsfähigen Systemen beeinträchtigen. Teste den Schmerz-Effekt auf verschiedenen Geräten, um sicherzustellen, dass die Framerate nicht erheblich sinkt, wenn der Effekt ausgelöst wird.
- **Trigger-Zone für den Schmerz-Effekt optimieren**:
  - Stelle sicher, dass der Collider, der den Schmerz-Effekt auslöst, korrekt positioniert und dimensioniert ist. Ein zu kleiner Trigger kann dazu führen, dass der Effekt nicht zuverlässig ausgelöst wird, während ein zu großer Trigger den Effekt unnötig oft auslöst.
- **Wartezeiten zwischen Effekten**:
  - Der Effekt des Schmerzes sollte nicht sofort wiederholt werden, wenn der Spieler schnell in und aus der Triggerzone geht. Stelle sicher, dass zwischen den Aktivierungen des Effekts eine sinnvolle Wartezeit besteht, um eine zu häufige und störende Wiederholung zu verhindern.
- **Kombination von visuellen und auditiven Effekten**:
  - Der Schmerz-Effekt kann durch zusätzliche auditive Effekte wie Schmerzschreie, Dämpfungsgeräusche oder schnelle, atemlose Atemgeräusche ergänzt werden, um das Schmerzempfinden zu verstärken. Stelle sicher, dass die audio-visuelle Kombination natürlich und nicht zu übertrieben wirkt.
- **Interaktive Trigger für den Schmerz-Effekt**:
  - Der Effekt kann auch mit anderen Gameplay-Mechaniken kombiniert werden, z. B. durch das Hinzufügen von Gesundheits- oder Schmerzleisten, die dynamisch den Grad des erlebten Schmerzes anzeigen. Auf diese Weise kannst du die Intensität des visuellen Effekts mit dem aktuellen Zustand des Spielers verknüpfen.
- **Funktionalität der Kamerawackeln-Animation testen**:
  - Die ShakeCamera-Funktion nutzt zufällige Werte für die Kamerabewegung. Achte darauf, dass der Effekt in allen Situationen gut aussieht und nicht zu abrupt oder unangenehm wird. Teste verschiedene shakeIntensity- und shakeDuration-Werte für die besten Ergebnisse.

4.2 Allgemeine Tipps & Warnungen

Bewusstsein für die Auswirkungen auf den Spieler:

- Psychologische Illusionen, insbesondere solche, die auf Emotionen wie Angst, Schmerz oder Unbehagen abzielen, können starke Reaktionen hervorrufen. Achte darauf, dass die Spielerfahrung immer auf das Spielerlebnis und nicht auf Überwältigung oder unangemessene Belastung abzielt. Zu starke Effekte können die Spieler vom Spiel ablenken oder sogar frustrieren.

Moderation und Ausgewogenheit:

- Setze psychologische Illusionen sparsam ein. Wenn diese zu häufig oder zu intensiv verwendet werden, können sie ihre Wirkung verlieren oder das Spielgefühl negativ beeinflussen. Achte darauf, dass die Illusionen in den Kontext der Spielmechanik und der Erzählung eingebunden sind und nicht willkürlich oder ausbeuterisch wirken.

Testen auf verschiedenen Geräten und in unterschiedlichen Szenarien:

- Psychologische Illusionen sollten auf verschiedenen Bildschirmgrößen, mit unterschiedlichen Grafikeinstellungen und unter verschiedenen Lichtverhältnissen getestet werden, um sicherzustellen, dass die Illusionen in allen Szenarien gut funktionieren. Insbesondere Post-Processing-Effekte und visuelle Verzerrungen können auf unterschiedlichen Geräten unterschiedlich wahrgenommen werden.

Spielerfeedback berücksichtigen:

- Nutze Nutzerfeedback, um die Auswirkungen der Illusionen zu verstehen. Einige Spieler könnten auf bestimmte psychologische Effekte empfindlicher reagieren, z. B. durch motion sickness bei schnellen Kamerawackeln oder unangenehmen visuellen Störungen. Gib den Spielern die Möglichkeit, diese Effekte zu deaktivieren oder zu mildern, um die Zugänglichkeit zu verbessern.

Vermeide Überanstrengung:

- Zu viele psychologische Illusionen auf einmal können die Spieler überfordern. Baue eine klare Balance zwischen intensiven und ruhigeren Momenten im Spiel ein. Der plötzliche Wechsel von extremen Zuständen (z. B. von Ruhe zu extremen visuellen Verzerrungen) kann zu einer unangenehmen Spielerfahrung führen.

Komplexität der Implementierung bedenken:

- Die Implementierung von psychologischen Illusionen kann technisch anspruchsvoll sein. Nutze Optimierungen und achte auf die Performance, besonders wenn du mehrere komplexe Effekte gleichzeitig anwendest. Achte darauf, dass die Nutzung von Post-Processing und visuellen Effekten nicht zu einer signifikanten Belastung für die Hardware des Spielers führt.

Vermeide visuelle Überstimulation:

- Zu viele visuelle Verzerrungen und starke Effekte können den Spieler überfordern und die Immersion stören. Setze Effekte gezielt und mit Bedacht ein, um den gewünschten psychologischen Effekt zu erzielen, ohne den Spieler zu überreizen.

Berücksichtigung der Zielgruppe:

- Unterschiedliche Zielgruppen reagieren unterschiedlich auf psychologische Illusionen. Spiele, die an jüngere Spieler oder empfindliche Nutzer gerichtet sind, sollten diese Effekte sorgfältig dosieren oder ganz vermeiden. Ältere oder erfahrene Spieler könnten in der Lage sein, intensivere Effekte besser zu verkraften.

Potenzielle gesundheitliche Auswirkungen:

- Bestimmte psychologische Illusionen, insbesondere starke visuelle Effekte, können bei einigen Spielern zu gesundheitlichen Beschwerden wie Kopfschmerzen oder Übelkeit führen. Stelle sicher, dass deine Illusionen für alle Spieler sicher sind und, wenn nötig, eine Warnung vor der Nutzung angezeigt wird.

Empfehlungen:

- Game Dev Geeks GAME-DEV, THE INDIE WAY., Screen Damage In Unity! (https://youtu.be/7MklBPjPmMg?si=Xzp_kiuuRAxSpr6u, 21.08.2023)
- Brackeys, EVERY Image Effect in Unity Explained - Post Processing v2 Tutorial (https://youtu.be/9tjYz6Ab0oc?si=_ZefclHk476KfjjS, 05.07.2020)

4.3 Verwendungszweck

Steigerung der Immersion:

- Psychologische Illusionen können helfen, die Immersion eines Spiels zu intensivieren, indem sie die Wahrnehmung des Spielers beeinflussen. Durch die gezielte Manipulation von Wahrnehmung, Raum und Zeit können die Spieler tiefer in die Spielwelt eintauchen, was besonders in narrativen oder atmosphärischen Spielen von großer Bedeutung ist.

Förderung emotionaler Reaktionen:

- Illusionen können so eingesetzt werden, dass sie starke emotionale Reaktionen wie Angst, Spannung oder Freude erzeugen. In Horror- oder Thriller-Spielen sind diese Effekte besonders nützlich, um ein Gefühl von Unbehagen oder Bedrohung zu erzeugen, während sie in anderen Genres genutzt werden können, um die Freude an Entdeckungen oder Erfolgserlebnissen zu verstärken.

Verstärkung des Erzählens:

- Psychologische Illusionen können als erzählerisches Werkzeug eingesetzt werden, um die Geschichte eines Spiels zu unterstützen. Sie können als Metaphern oder visuelle Darstellungen innerer Konflikte oder emotionaler Zustände der Charaktere fungieren, die das narrative Erlebnis bereichern und den Spieler stärker in das Geschehen einbinden.

Spielmechanische Verstärkung:

- In bestimmten Spielmechaniken können psychologische Illusionen als Werkzeug eingesetzt werden, um das Spielerlebnis herausfordernder zu gestalten. Beispielsweise können Illusionen in Form von Veränderungen der Perspektive, Verzerrungen der Raumwahrnehmung oder des Kälte- bzw. Schmerzempfindens genutzt werden, um dem Spieler neue Herausforderungen zu präsentieren, die seine Wahrnehmung und Reaktionen auf die Spielwelt verändern.

Stimulation von Erinnerungen und Problemlösungsfähigkeiten:

- Illusionen, die die Wahrnehmung der Zeit, der Raumverhältnisse oder der visuellen Informationen manipulieren, können die Problemlösungsfähigkeiten des Spielers auf die Probe stellen. In Puzzle- oder Abenteuerspielen können solche Effekte dazu verwendet

werden, den Spieler in eine neue Denkweise zu versetzen, die ihm hilft, neue Lösungen zu finden.

Erzeugung von Überraschungseffekten:

- Der Überraschungseffekt ist ein weiterer vielseitiger Verwendungszweck psychologischer Illusionen. Durch die Manipulation der Wahrnehmung können Entwickler Unerwartetes erzeugen, indem sie das, was der Spieler für Realität hält, in Frage stellen. Dies kann dazu beitragen, das Gameplay aufregender und dynamischer zu gestalten.

Erweiterung des Spielgenres:

- Psychologische Illusionen bieten eine Möglichkeit, Genregrenzen zu überschreiten und traditionelle Spielmechaniken zu erweitern. Sie können in ungewöhnlichen Genre-Kombinationen verwendet werden, um neue Spielerfahrungen zu schaffen, wie etwa in interaktiven Geschichten, Escape-Room-Spielen oder sogar edukativen Spielen, um ein noch tieferes Spielerlebnis zu bieten.

Integration in die Atmosphäre und Weltgestaltung:

- Die psychologische Manipulation von Licht, Schatten und Farbe kann auch eingesetzt werden, um eine Atmosphäre zu schaffen, die mit der Spielwelt harmoniert. In Fantasy- oder Sci-Fi-Spielen können diese Illusionen genutzt werden, um surreale Welten zu erschaffen, die die Grenzen der Realität überschreiten und den Spieler immer wieder überraschen.

Optimierung der Spielererfahrung:

- Psychologische Illusionen können auch die allgemeine Spielererfahrung optimieren, indem sie das Tempo und die Intensität von Ereignissen in einem Spiel steuern. So kann eine ruhige, langsame Erkundungsphase plötzlich in eine intensive, temporeiche Herausforderung umgewandelt werden, die den Spieler fesselt und motiviert.

Quellenverzeichnis

Icons:

[1] Boxicons. Cog. Verfügbar unter: https://boxicons.com/?query=cog (Stand: 10.03.2025)

[2] Boxicons. Show. Verfügbar unter: https://boxicons.com/?query=show (Stand: 10.03.2025)

[3] Boxicons. Cube. https://boxicons.com/?query=cube (Stand: 10.03.2025)

Tutorials:

[4] RumpledCode. (02.12.2020). Superliminal core scaling mechanic - Unity Tutorial [Video]. YouTube. Verfügbar unter: https://youtu.be/BNYy6O_Uy2Y?si=bxZtUIwBPTUSQAx9 (Stand: 10.03.2025)

[5] Mujj. (01.05.2021). Remaking Superliminal's Mindbending Illusions… [Video]. YouTube . Verfügbar unter: https://youtu.be/QogeuKwlPNs?si=rZbs1A97GCz9WILh (Stand: 10.03.2025)

[6] SpeedTutor. (26.03.2022). PICKUP & DROP Physics Objects in Unity (Like PORTAL) [Video]. YouTube. Verfügbar unter: https://youtu.be/6bFCQqabfzo?si=329kJEtn8yTkZZZw (Stand: 10.03.2025)

[7] experimentis. Irre Täuschung: Der Ames-Raum [Webseite]. experimentis. Verfügbar unter: https://www.experimentis.de/experimente-versuche/optik-akustik/ames-room-ames-raum-basteln-adelbert-ames/ (Stand: 10.03.2025)

[8] d'Art of Science. (23.12.2016). AMES ROOM Illusion | REVEALED | dArtofScience [Video]. YouTube. Verfügbar unter: https://youtu.be/lrVWnDPybyc?si=usQQL2AuTc0s8ezo (Stand: 10.03.2025)

[9] UnityDude. (02.08.2024). Crafting mirrors in Unity : Reflections Tutorial [Video]. YouTube. Verfügbar unter: https://youtu.be/_jBWm0KM2ec?si=R_q3Q2Icjz5jONbY (Stand: 10.03.2025)

[10] Game Dev Guru. (25.12.2022). Unity - How to make a mirror in Unity [2023 UPDATED] [Video]. YouTube. Verfügbar unter: https://youtu.be/RzT4m1-ItvU?si=yADNmgperDWvDvwN (Stand: 10.03.2025)

[11] Brackeys. (07.01.2018). Smooth PORTALS in Unity [Video]. YouTube. Verfügbar unter: https://youtu.be/cuQao3hEKfs?si=AzIBX-Jq7jEvdrL0 (Stand: 10.03.2025)

[12] Dani Krossing. (07.12.2021). HOW TO TELEPORT OBJECTS IN UNITY 🎮 | Teleport Player and GameObjects in Unity | Unity Tutorial [Video]. YouTube. Verfügbar unter: https://youtu.be/xmhm5jGwonc?si=idW07fOvMZVZNKKQ (Stand: 10.03.2025)

[13] passivestar. (26.06.2022). Teleporting - Unity Tutorial [Video]. YouTube. Verfügbar unter: https://youtu.be/I7M8T3qU-_E?si=tlLdLHbhWKWbJjxt (Stand: 10.03.2025)

[14] Daniel Ilett. (05.01.2022). Impossible Geometry with Stencil Shaders in Unity URP [Video]. YouTube. Verfügbar unter: https://youtu.be/EzM8LGzMjmc?si=TrL2_s_IrOSeYoyy (Stand: 10.03.2025)

[15] Karl Henkel. (12.03.2015). ProBuilder Basic Room Tutorial [Video]. YouTube. Verfügbar unter: https://youtu.be/ll7PpGXuHwI?si=nOhpILyrrrQ6udbS (Stand: 10.03.2025)

[16] DarkDax. (28.02.2023). 12 Level Design Tips to Optimize Your Indie Game You Should Know [Video]. YouTube. Verfügbar unter: https://youtu.be/H-2IYnM3x8g?si=hlcasGimHH3P9LL4

[17] Brackyt. (09.02.2020). Room Changing Effect - Unity [Video]. YouTube. Verfügbar unter: https://youtu.be/CWNAGm36_gg?si=113sKloru5Pdl8mp (Stand: 10.03.2025)

[18] Jayanam. (05.05.2018). Unity Beginner Tutorial : Triggers [Video]. YouTube. Verfügbar unter: https://youtu.be/JHZN_vMqqr4?si=MT6je6KKcSTtlLvj (Stand: 10.03.2025)

[19] LlamAcademy. (09.11.2021). Doors in Unity - Sliding and Rotating Doors | Make Them Automatically Open/Close or Usable [Video]. YouTube. Verfügbar unter: https://youtu.be/cPltQK5LlGE?si=2LeFOMMqxV-A_9IZ (Stand: 10.03.2025)

[20] Brackeys.(24.01.2018). LIGHTING in Unity [Video]. YouTube. Verfügbar unter: https://youtu.be/VnG2gOKV9dw?si=AwERNk2fmTYlPDyT (Stand: 10.03.2025)

[21] Fist Full of Shrimp. (05.01.2024). Light Up Your Game : Unity Lighting Essentials! [Video]. YouTube. Verfügbar unter: https://youtu.be/5rxMdiCkQGk?si=ULPkHYzYQ6RezQqf (Stand: 10.03.2025)

[22] LMHPOLY. (11.11.2020). Unity 2020 Tutorial - Lighting And Post-Processing Low Poly Scene [Video]. YouTube. Verfügbar unter: https://youtu.be/uwYf1yMPuvk?si=jGD1-BiS0WaKANqm (Stand: 10.03.2025)

[23] GDT Solutions ES. (16.11.2020). How to replace DEFAULT SKYBOX in Unity - Custom Skybox with 360 TEXTURE [Video]. YouTube. Verfügbar unter: https://youtu.be/ckgfQdaEdwk?si=8k9HI1A3_QWbEl_C (Stand: 10.03.2025)

[24] Unity Ace. (04.01.2022). How to Add Fog in Unity [Video]. YouTube. Verfügbar unter: https://youtu.be/RBI9hQWD_xY?si=Tze9IXUacXnWTydm (Stand: 10.03.2025)

[25] Benjamin Swee - Custom Unity Shaders. (23.04.2022). Custom Unity Distortion Shader Using Post Processing [Video]. YouTube. Verfügbar unter: https://youtu.be/YSAqivxqC3w?si=JLe4Emr6uiVZtED2 (Stand: 10.03.2025)

[26] Mech Mind Games. (21.04.2021). HOW TO MAKE DISTORTION/HEAT EFFECT FOR 2D & 3D - SIMPLE SHADER GRAPHS - Unity 2019 LTS [Video]. YouTube. Verfügbar unter: https://youtu.be/rSiYL04H1jE?si=weaL2QSog8ioyqc8 (Stand: 10.03.2025)

[27] Gabriel Aguiar Prod.. (09.02.2021). Unity VFX Graph - Heat Distortion Effect Tutorial [Video]. YouTube. Verfügbar unter: https://youtu.be/CXCyVDEplyM?si=aXpGsZ4pS9rgaOLU (Stand: 10.03.2025)

[28] PolyToots. (19.08.2020). Unity/Amplify: Heat Haze Distortion, Boiling Bubbles, Fire/Smoke and God Rays! [Video]. YouTube. Verfügbar unter: https://youtu.be/jFkhFuc0H4k?si=E28PBD0T4WEKhhmq (Stand: 10.03.2025)

[29] Mirza. (08.01.2022). Snow Particle System Tutorial for Unity in just 2 Minutes [Video]. YouTube. Verfügbar unter: https://youtu.be/T_8cPlpTr90?si=P-nddWs7xdrL9hfI (Stand: 10.03.2025)

Assets:

[30] Yi Zhen. (letzte Veröffentlichung: 5.10.2024). Fantasy Mirror Primer [Asset]. Unity Asset Store. Verfügbar unter: https://assetstore.unity.com/packages/vfx/shaders/fantasy-mirror-primer-189252 (Stand: 10.03.2025)

[31] Brackeys. (01.05.2018). ScreenCutoutShader.shader [Shader]. GitHub. Verfügbar unter: https://github.com/Brackeys/Portal-In-Unity/blob/master/Portal/Assets/ScreenCutoutShader.shader (Stand: 10.03.2025)

[32] rpgwhiteloc*k.* (letzte Veröffentlichung: 20.07.2024). AllSky Free - 10 Sky / Skybox Set [Asset]. Unity Asset Store. Verfügbar unter: https://assetstore.unity.com/packages/2d/textures-materials/sky/allsky-free-10-sky-skybox-set-146014 (Stand: 10.03.2025)

[33] TriForge Assets. (letzte Veröffentlichung: 23.07.2019). Fantasy Forest Environment - Free Demo [Asset]. Unity Asset Store. Verfügbar unter: https://assetstore.unity.com/packages/3d/environments/fantasy/fantasy-forest-environment-free-demo-35361 (Stand: 10.03.2025)

[34] Barking Dog. (letzte Veröffentlichung: 22.07.2024). 3D Free Modular Kit [Asset]. Unity Asset Store. Verfügbar unter: https://assetstore.unity.com/packages/3d/environments/3d-free-modular-kit-85732 (Stand: 10.03.2025)

[35] WALLCOEUR. (letzte Veröffentlichung: 20.12.2024). VFX URP - Fire Package [Asset]. Unity Asset Store. Verfügbar unter: https://assetstore.unity.com/packages/vfx/particles/fire-explosions/vfx-urp-fire-package-305098 (Stand: 10.03.2025)

[36] Simon Serge Pasi. (letzte Veröffentlichung: 06.07.2021). Mini First Person Controller [Asset]. Unity Asset Store. Verfügbar unter:

https://assetstore.unity.com/packages/tools/input-management/mini-first-person-controller-174710 (Stand: 10.03.2025)

Abbildungsverzeichnis

Abbildung 1: Perspektivische Größenveränderung.... 4
Abbildung 2: Ames-Raum....9
Abbildung 3: Spiegelreflexion.... 15
Abbildung 4: Teleportation im freien Fall....21
Abbildung 5: Nichteuklidische Portale.... 23
Abbildung 6: Raumaufbau.... 28
Abbildung 7: Trigger Buttons.... 31
Abbildung 8: Trigger gesteuerte Tür.... 38
Abbildung 9: Tiefenwirkung.... 44
Abbildung 10: Ebbinghaus-Illusion.... 48
Abbildung 11: Größen Illusion durch Positionierung....50
Abbildung 12: Dynamische Größenveränderung....52
Abbildung 13: Einfluss von Licht und Farbe auf die Größe.... 55
Abbildung 14: Tunnelblick....59
Abbildung 15: Wärmeempfindung.... 61
Abbildung 16: Hitzewellen.... 63
Abbildung 17: Kälteempfindung....67
Abbildung 18: Size over Lifetime Kurve....69
Abbildung 19: Schmerzempfinden Overlay.... 71
Abbildung 20: Schmerzempfinden Post Processing....74